الجمهوريـة الجزائـريـة الديمقراطيـة الشعبيـة
République Algérienne Démocratique et Populaire
وزارة التعليـم العالـي والبحـث العلمـي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Nº Réf :……

Centre Universitaire
Abd Elhafid Boussouf Mila

Institut des Sciences et Technologie **Département de Mathématiques et Informatique**

Mémoire préparé en vue de l'obtention du diplôme de Master

En : Informatique
Spécialité: Sciences et Technologies de l'Information et de la Communication (STIC)

Development of a video game using multi-agent modelling

Préparé par : Brichen DiaeDdine

Soutenue devant le jury

Encadré par : Dib Abderrahim.	**MAA C.U.Abd Elhafid Boussouf**
Président : Samir Selmane.	**MAA C.U.Abd Elhafid Boussouf**
Examinateur : Djaaboub Salim.	**MCB C.U.Abd Elhafid Boussouf**

Année Universitaire : 2021/2022

Acknowledgements

Above of all, I thank Almighty **God** for his help and guidance in doing this work

the first person I would like to thank is my framer **Mr. ∗Dib Abderrahim∗**

I would also like to thank my dear parents, for all their sacrifices, their support,and their prayers for me.

I won't forget my brother and sister also for being there for me.

my warmest thanks go to all the teachers who followed me during my five years of studies at the university.

Finally, I would like to thank all those who have contributed in any way to the realization of this work.

Abstract

The last decade has seen Artificial Intelligence (AI) seep into the game development industry, Artificial intelligence is used in video games to create believable characters that challenge the player and enrich the playing experience. The adventure genre is one of the types of games where it is fundamental to have non-player characters with the ability to perceive the environment, detect some of the player character's actions and react to them with astute strategies.

The goal of this project was to develop a game with enemy characters that would cooperate to defeat the player. In the final game, the enemies patrol the game's world trying to locate the player character. The findings of an enemy are immediately communicated to the rest of agents, and the player character is chased and attacked by the enemies after being spotted. Agents pursue the player character and use close combat moves.

The 3d game objects was developed with the Blender software, The game was developed with the Unreal Engine 4 game engine. The reasoning and decision making of enemies was modelled using behaviour trees. The implemented system makes agents capable of perceiving visual and auditory stimuli and communicating information to other agents, who compare it with their own knowledge to take independent decisions.

The whole set of enemies form an artificial intelligence-based multi-agent system and cooperate to achieve the shared goal of defeating the player in the shortest time possible. The ability of the numerous enemies to spot the player character and detect sounds of players.Unlike them, player try struggle to out of the perception system, and Avoid contact with them.

ملخص

شهد العقد الماضي تسرب الذكاء الاصطناعي (AI) إلى صناعة تطوير الألعاب ، ويستخدم الذكاء الاصطناعي في ألعاب الفيديو لإنشاء شخصيات يمكن تصديقها تتحدى اللاعب وتثري تجربة اللعب. نوع المغامرة هو أحد أنواع الألعاب حيث من الأساسي أن يكون لديك شخصيات غير لاعب لديها القدرة على إدراك البيئة ، واكتشاف بعض تصرفات شخصية اللاعب والرد عليها باستراتيجيات ذكية.

الهدف من هذا المشروع هو تطوير لعبة تحتوي شخصيات تمثل الاعداء يمكنهم التتعاون لهزيمة اللاعب. في اللعبة النهائية ، يقوم الأعداء بدوريات في عالم اللعبة في محاولة لتحديد شخصية اللاعب. يتم إبلاغ نتائج العدو عند توفرها على الفور إلى بقية العملاء ، ويتم ملاحقة شخصية اللاعب ومهاجمته من قبل الأعداء بعد أن يتم رصده. يتابع العملاء شخصية اللاعب ويستخدمون حركات قتالية قريبة.

تم تطوير كائنات اللعبة ثلاثية الأبعاد باستخدام برنامج Blender. و تم تطوير اللعبة باستخدام محرك اللعبة Unreal Engine 4. تم تصميم التفكير واتخاذ القرار للأعداء باستخدام أشجار السلوك حيث يجعل النظام المنفذ الوكلاء قادرين على إدراك المحفزات البصرية والسمعية ونقل المعلومات إلى الوكلاء الآخرين ، الذين يقارنونها بمعرفتهم لاتخاذ قرارات مستقلة.

تشكل المجموعة الكاملة من الأعداء نظامًا متعدد العوامل قائمًا على الذكاء الاصطناعي والتعاون لتحقيق الهدف المشترك المتمثل في هزيمة اللاعب في أقصر وقت ممكن باستخدام قدرة الأعداء على اكتشاف شخصية اللاعب. على عكسهم ، يحاول اللاعب الكفاح للخروج من نظام الإدراك ، وتجنب الاتصال بهم.

Résumé

La dernière décennie a vu l'intelligence artificielle (IA) s'infiltrer dans l'industrie du développement de jeux. L'intelligence artificielle est utilisée dans les jeux vidéo pour créer des personnages crédibles qui défient le joueur et enrichissent l'expérience de jeu. Le genre aventure est l'un des types de jeux où il est fondamental d'avoir des personnages non-joueurs capables de percevoir l'environnement, de détecter certaines actions du personnage joueur et d'y réagir avec des stratégies astucieuses.

Le but de ce projet était de développer un jeu avec des personnages ennemis qui coopéreraient pour vaincre le joueur. Dans le jeu final, les ennemis patrouillent dans le monde du jeu en essayant de localiser le personnage du joueur. Les découvertes d'un ennemi sont immédiatement communiquées au reste des agents, et le personnage du joueur est poursuivi et attaqué par les ennemis après avoir été repéré. Les agents poursuivent le personnage du joueur et utilisent des mouvements de combat rapproché.

Les objets de jeu 3D ont été développés avec le logiciel Blender, Le jeu a été développé avec le moteur de jeu Unreal Engine 4. Le raisonnement et la prise de décision des ennemis ont été modélisés à l'aide d'arbres de comportement. Le système mis en œuvre rend les agents capables de percevoir des stimuli visuels et auditifs et de communiquer des informations à d'autres agents, qui les comparent à leurs propres connaissances pour prendre des décisions indépendantes.

L'ensemble des ennemis forme un système multi-agents basé sur l'intelligence artificielle et coopère pour atteindre l'objectif commun de vaincre le joueur dans les plus brefs délais. La capacité des nombreux ennemis à repérer le personnage du joueur et à détecter les sons des joueurs. Contrairement à eux, le joueur essaie de lutter pour sortir du système de perception et évite tout contact avec eux .

Contents

List of Figures

List of abbreviations and acronyms

AI Artificial Intelligence

MAS Multi-Agent Systems

BT Behavior Tree

FSM Finite State Machines

GA Genetic Algorithms

QA Quality Assurance

EQS Environment Query System

GDD Game Design Document

RL Reinforcement Learning

NPC Non Player Character

HUD Heads-Up Display

GUI Graphical User Interface

PBR Physically Based Rendering

UE Unreal Engine

General introduction

Game evolution/design is the procedure of producing a video game. Game evolution has evolved in recent years. No longer does making a game involve writing simple lines of code by an average programmer.

Now it requires a team of specialists in disciplines such as Fine art, Graphics Modeling and Design, Software Programming, Music, Network Programming, AI Programming, and so on. Games have protracted happened to be an accepted field of Artificial intelligence(AI) research, which is for a respectable motive. They are problematic nevertheless stress-free to validate, hence making it feasible to establish novel AI techniques, compute in what way they are functioning, and display that machines are qualified of extraordinary usually alleged to necessitate cleverness exclusive of placing person breathes or property at jeopardy AI is the field within Computer Science that seeks to explain and to emulate some or all aspects of human intelligence through mechanical or computational processes. Included among these aspects of intelligence are the ability to interact with the environment through sensory means and the ability to make decisions in unforeseen circumstances without human intervention. Typical AI research areas include game playing, natural language understanding, and synthesis, computer vision, problem-solving, learning, and robotics.

Over the years, there has been an increase in the need for AI in Game Development. Implementing AI in a game will give the users the illusion that they are playing intelligent agents. From the definition of intelligent agents in Artificial Intelligence, we can say an intelligent agent is anything that can perceive/observe its immediate environment and take action concerning its observation, hence we can say an intelligent gaming agent is capable of learning/observing what goes on in the gaming environment and also act on its observation.This research study focuses majorly on how AI is implemented in Game Development and it was implemented using C++ language.

Problem

The enemy system is considered one of the most sensitive systems in the game because the interaction of enemies with the player reflects the extent of the fun or boredom within the game. Whereas, the more realistic the interaction, the greater the challenge within the game, and therefore various ways to play. The problem at hand is how to create a realistic, effective, enjoyable, and scalable enemy system.

Objectives

The objective of this project is to develop a game with an AI-based multi-agent system that would represent the behavior of enemies. Planned development tasks should be within the expected timescale.

Organize thesis

This thesis is divided into four chapters:

The first chapter: "Game Development", Bibliographical research that includes an overview of the basic theoretical and applied concepts of game development.

The second chapter: " Artificial Intelligence In Game Development ", Artificial intelligence in games and the various (AI) techniques used in game development.

The third chapter: " System conception ", Describes the architecture of the system to be implemented

The fourth chapter: " System Implementation ", It shows the tools and programming languages used in creating the game and displays the final results.

Chapter 1

Game development

1.1 Introduction

With the speedy development of computer technology, the importance of software program engineering in our everyday lives is increasing. It impacts each element of our lives today, such as working, living, learning, and education. A new and famous mode of enjoyment and a vital generation of software program video games, that have ended up becoming accepted by human beings of all ages. In today's culture, the generation is without difficulty reachable and has ended up extra convenient; increasingly more human beings want to play video games and also are turning into prompted to layout their very own video games. [1] defined "game is a software application in which one or more players make decisions by controlling game objects and resources, in the pursuit of its goal". video games are software programs established on hardware gadgets and online game consoles, computers, handheld gadgets, and Personal Digital Assistants (PDAs)[2]. Software video games have now end up a international innovative industry, however, due to the multidisciplinary games required, their improvement is a completely complicated task[4].

The multidisciplinary nature of the strategies that integrate sound, art, manipulate systems, synthetic intelligence (AI), and human factors, additionally, make the software program recreation improvement exercise unique from conventional software program improvement. However, regardless of the excessive complexity of the software program engineering improvement process, the games enterprise is making billions of bucks in income and growing many hours of fun . The software program recreation marketplace at some stage in the sector has grown through over 7–8 % yearly and has reached income of around $5.5 billion in 2015 . Newzoo Game Market has additionally suggested that the sector-huge virtual recreation marketplace will reach $113 billion through 2018.

The creation of any game involves cross-functional brigades including designers, software inventors, musicians, scriptwriters, and numerous others. Also, Entertainment Software Association reports stressed the rearmost trends in the software game industry. Thus, game development careers have presently come largely grueling, dynamic, creative, and profitable . The capability to handle complex development tasks and achieve profitability doesn't be by chance, but rather a common set of good practices must be espoused

to achieve these pretensions. The game assiduity can follow the good and proven practices of traditional software engineering, but only a clear understanding of these processes can enhance the complex game development engineering process.

1.2 Game types summary

Utmost ultramodern video games can be assigned to a particular genre, or classified as a hybrid of two or more genres. These genres have come about over the years, frequently as a result of trial and error, but more frequently as an evolution.[6] The following is a description of some important genres and the games that either introduced or popularized them.

Platformer The original platform games involved the character running and jumping on a side-scrolling playing field. While the definition has been expanded now to include 3D playing fields, the genre is still fairly true to its roots. Some of the most famous platformers have been Super Mario Bros, Crash Bandicoot, Sonic Mania, Cuphead, and It Takes Two.[1]

First-Person Shooter The first-person shooter is an action game that places the player "behind the eyes' of the game character. In this game, the player can wield a variety of weapons and fight enemies by shooting them. [1]This genre is famous for games like Wolfen stein 3D, Doom, call of duty, and battlefield.

Action The action game is the superset of many other genres. First-person shooters, combat simulations, fighting games, even platform games are all parts of the action genre.[6] Games in the action genre are typified by fast-paced combat and movement. Some of the old examples of video games such as Spacewar, Pong, and Space Invaders defined the genre and also was the cause of his success.

Adventure In the adventure game genre, there have been two important subgenres: the text-based adventure and the graphical adventure. For text-based breakouts, one needs to look no further than Zork by Infocom. On the graphical adventure side, one of the series that defined the genre was the King's Quest series from Roberta Williams at Sierra.

Action-Adventure Action-adventure games are similar to adventure games but incorporate action elements. Nintendo's The Legend of Zelda was the first breakout hit of the genre, but there have been many more since.[1] Recent games like Control, Assassin's Creed, and Resident Evil continue the tradition of action with strong puzzle-solving.

Fighting In fighting games, the player fights other players or the computer(AI). These games originated in the arcades, where players could signify their intent to challenge one another by placing quarters on the top of the cabinet. Super Smash Bros is one of

the most famous games in the genre, allowing players to fight side by side. Tekken, Street Fighter, and Mortal Kombat are two of the most famous fighting games in which players choose characters and fight against each other (called a versus fighter).[2][1]

Real-Time Strategy (RTS) In a typical Real-Time Strategy (RTS), the goal is for the player to collect resources, build an army, and control his units to attack the enemy. The action in these games is fairly fast-paced and because of the continuous play, strategic decisions must be made quickly. While The Ancient Art of War in 1984's and Herzog Zwei in 1989's were early examples of the genre, the games that popularized it were Westwood's Dune 2 and Command and Conquer, and Blizzard Warcraft.

Turn-Based Strategy These games are similar to real-time strategy games (indeed, they were the precursors to them), but the players take turns in which they make their moves. For example, almost all board games (like Chess and Checkers) are turn-based. In the era of the RTS, turn-based games are less frequently made, but there are some notable games in the genre, X-COM, namely Civilization, Master of Orion, and. Jagged Alliance.

Stealth Stealth games (sometimes called sneakers) are characterized by their focus on subterfuge and their planned-out, deliberate gameplay. They are by and large similar to first-person or third-person shooters but are less action-oriented and more methodical. The first stealth game was the original Metal Gear in 1987, but other notable stealth games include the Thief series, the Metal Gear series, and the Splinter Cell series.

Role-Playing Game (RPG) The video game version of pen and paper games like Dungeons & Dragons differs from its tabletop counterpart generally in its ability to create a world that doesn't require imagination. Most differentiations from the system are hybrids with other genres. Some of the most famous RPGs to grace computer and TV screens are the Final Fantasy series, the Baldur's Gate series, and Wasteland.[1]

Massively Multiplayer Online Role-Playing Game (MMORPG) The MMORPG or MMO is a role-playing game set in a persistent virtual world populated by thousands of players simultaneously connected over the Internet. The MMO was predated by text-based games called Multi-User Dungeons/Dimensions (MUDs), but in modern times it is largely graphical. In the games, the player is represented by a character called an avatar. The first accident MMO was Meridian 59 in 1996. The first current implementation, however, was Ultima Online in 1997. World of Warcraft is currently the king of the genre with more than 117 million subscribers.

Serious The serious game genre has emerged in the past years as a cheaper and more entertaining way of teaching real-world events or processes to adults. These video games are generally privately funded for particular uses, with the U.S. authorities and clinical experts being the biggest customers. For example, games builders can expand education simulators quite cheaply, even by infusing the simulation with leisure fees. [2] The laugh

fee is essential in order that customers are stimulated to replay the sport regularly and for that reason come be higher trained The Game Developers Conference has recognized the strong interest in serious games and in 2004 added a two-day Serious Games Summit as part of its annual event, focusing on "the intersection of games, learning, policy, and management." [GDC][1].

Simulation Simulation games are based on the simulation of a system. This system may be whatever from the workings and economic system of the railroads (which includes in Railroad Tycoon) to a fight state of affairs in which the participant controls huge moves of troops or maybe single fighter craft. SimCity is one of the breakout simulation games, allowing you to micro-manage a city. Wing Commander and X-Wing are two of the defining space combat simulation games. Microsoft Flight Simulator is one of the most famous airplanes simulation games. IIn current years, The Sims is one of the extra famous video games withinside the genre, with its complicated simulation of human lifestyles and social interactions.

Racing Racing games involve competing in a race in vehicles ranging from race cars to motorcycles to go-karts. This style is has a touch specific from others in that the video games basically try and re-create as best they could a real-world activity. The first appearance of this type of racing game was Pole Position from Atari. [1][2][6]

Sports The sports activities sport style covers a myriad of video games that simulate the sporting experience. As with racing video games, sports activities video games are in the main try to re-create the complicated interactions in an actual sport. Some games series adopt this style like John Madden Football and Tiger Woods Golf.

Rhythm Rhythm games gauge a player's success based on his ability to trigger the controls in time to the beat of the music. Some video games, along with Konami s Dance Dance Revolution (DDR), require the participant to step on ground pads in time to music, even as Nintendo's Donkey Konga for the Nintendo GameCube comes with a specialized bongo drum controller—even though now no longer all rhythm video games require specialized controllers. For example, PaRappa the Rapper seems because the first sizable rhythm game, performed on the PlayStation in 1996, and it most effectively required the usual controller. Currently, the Guitar Hero and Rock Band franchises are answerable for the exceptional reputation of this form of the game.

Puzzle Puzzle video games integrate factors of sample matching, logic, strategy, and luck—frequently with a time element. Tetris is the most famous puzzle game ever and serves as a first-rate instance of the style with its frenetic pattern-matching action.[6]

Mini-Games Mini-games are commonly short, easy video games that exist inside a bigger conventional game. They are now and again used as praise for finishing a venture or unlocked via way of means of coming across a secret. Alternately, the bigger game may be a skinny veil for a set of mini-video games, as withinside the Mario Party collection

or the Wario Ware collection. The Wario Ware collection is a unique account that every tide carries extra than one hundred video games, with each lasting only several seconds or minutes. Many video games on the Internet used for marketing functions may also be defined as mini-video games.

Traditional Traditional video games encompass automated variations of card video games and board video games. The first conventional game applied on a laptop display screen changed into Noughts and Crosses (tic-tac-toe) via way of means of A. S. Douglas at the University of Cambridge in 1952. Throughout the years, chess has long been a staple of conventional video games, with Chessmaster being the maximum diagnosed collection. In 1988, Interplay advanced Battle Chess, which changed into simply regular chess, however, whilst every piece took another, there has been a unique (and regularly humorous) animation of the "battle." Sierra's Hoyle collection is one of the maximum devoted efforts to deliver conventional video games to a computer format, with its trustworthy translations of the card, board, word, table, and puzzle games[2].

Educational Educational video games are designed to teach grade-school ideas to children and teens in a wonderful manner. notable educational game was Oregon Trail, at the beginning, designed in 1971 for teletype machines at Carleton College, however made famous withinside the Eighties and Ninetiesby a version running on Apple computers systems in public schools. Other incredible video games on this style encompass the Carmen Sandiego collection and Mavis Beacon Teaches Typing.[1]

Survival Horror Survival horror is a subgenre of action-adventure and first-person shooter games. Typically, they contain exploring deserted homes or cities wherein numerous monsters and undead creatures lurk. The survival factors are focused on in no way giving the participant pretty sufficient bullets or health, as a result growing the tension. The horror issue defines the topic and pacing, generally directing the participant to discover quiet places, deserted, bloodstained hallways until a monster comes crashing through a window, or a seemingly lifeless corpse begins to stir. Players are often started and can become visibly shaken from the experience, much like a good horror movie. Alone in the Dark is recognized as the first in the genre in 1992, Resident Evil popularized the "survival horror" and Set standards for later games in 1996. [6]

1.3 Varieties of player arrangements

In narrative arts (literature, theater, movies, etc.), various types of quarrel, related to dramatic struggle, are categorized—man vs. man, man vs. nature, etc. In games, these narrative conflicts can still exist. At a far more specific level, games offer all sorts of other kinds of conflict such as different styles of fighting within a single game. So the word "conflict" can get overloaded quickly. We will use player arrangement for those configurations of player conflict (Figure 1). In the book Game Design Workshop, Tracy Fullerton uses the term interaction patterns and adapts a scheme that offers a nice illustration of the various forms these game conflicts can take.The following are some typical

player arrangements:
- Single-player—player contends with the game system.
- Player vs. player—two players contend with each other.
- Multilateral competition—three or more players contend with each other.
- Team competition—two groups compete with each other.
- Multilateral team competition—three or more groups compete.
- Unilateral competition—two or more players compete with one player.
- Multiple individual vs. game—multiple players compete against the system.
- Cooperative—two or more cooperate against the game system. .

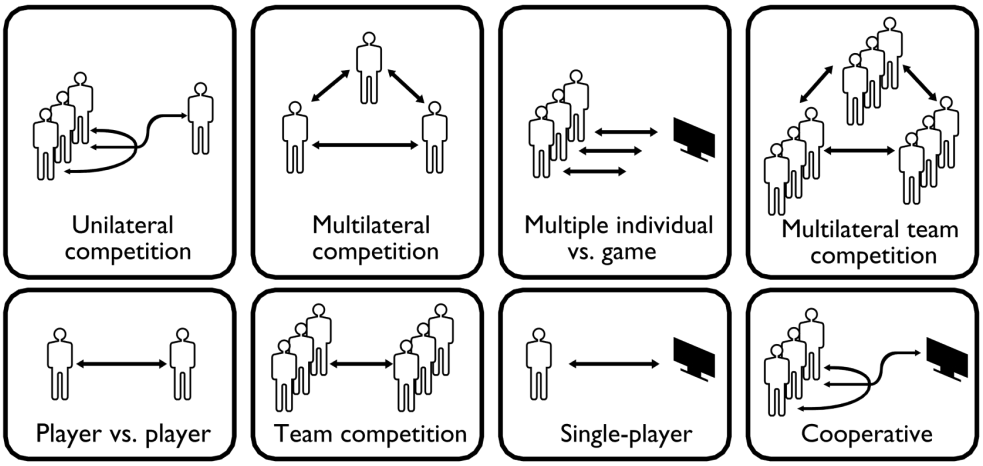


Figure 1.1: Varieties of player arrangements

1.4 The game development process

One of the main studiousness in the game development process is that developers need to follow best practices and procedures from the software engineering discipline to develop good quality games. The game development process comprises four main phases: concept, pre-production, production, and post-production. It consists of various activities such as synopsis, scriptwriting, background research, visualization and concept art, level and interaction design, animation, programming, media editing, integration, testing, and publishing.[6] Software games are also characterized based on the category into which they fall, which is called the genre of the game(We talked about it before). Each genre has its requirements which must be taken into consideration during the pre-production phase. For this reason, software game development is considered a complex process that involves multidisciplinary collaborative team efforts and processes (including modeling (3D/2D art), sound, gameplay, artificial intelligence, control systems, human factors ..etc) to develop a creative product. Fundamentally, game development is a form of the software development process with several additional requirements such as creative design, artistic aspects, and visual presentation[10].

Software game development additionally necessitates a number of skills that encompass project management, design, improvement, and asset creation. It additionally consists of group contributors from heterogeneous Specialties, [4]e.g., game designers, artists, programmers, and software program builders. Knowledge of great practices for recreation development may be very essential and has come to be decisive in maintaining the increase of the software program recreation industry. Finally, this information will assist creation making accurate game development selections at the proper time. Achievement of these key elements from a developer's angle will make contributions to the knowledge of the contemporary development method The ensuing outcomes will assist programmers to enhance the game development method[7].

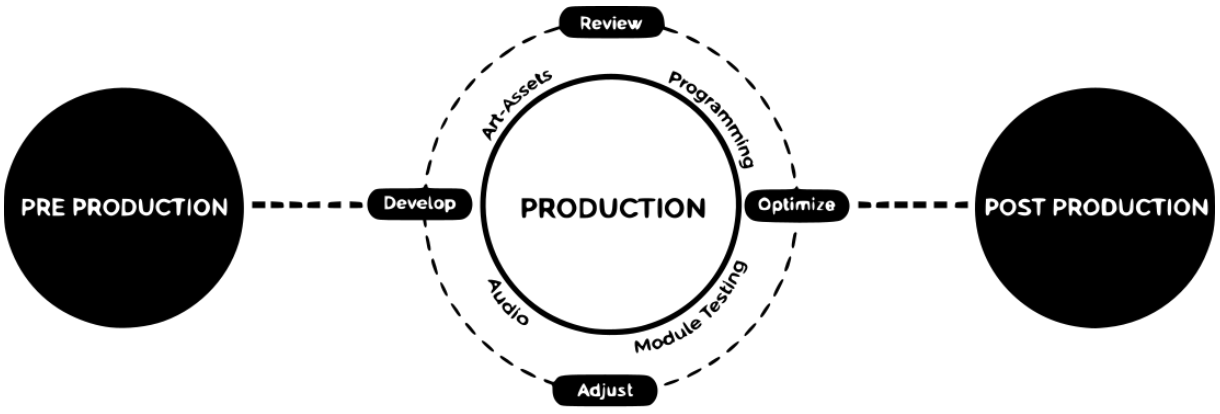


Figure 1.2: The game development process

.

1.4.1 Concept

The unique concept is only an easy concept for what the game may be about. For instance, an easy game idea may be to make a futuristic 3-d avenue racing sport with hovercrafts with a placing corresponding to the cutting-edge import tuner avenue racing scene of modern-day international. It also can be some thing as easy as making an action/adventure game where you're controlling a ninja.

The games concept also can begin as in reality trying to make a follow-up or sequel to an current title, a sport primarily based totally on current non-gaming characters, stories, or franchises - from different mediums along with tv, comedian books, board games, movies, folklore, or history - or a sport it is supposed to simulate some real-world experience, along with the case with sports, flight, or riding simulations. In those cases, the genesis of the sport's improvement can in reality be the agency determining that it desires to make a game that simulates the real-lifestyles[4].

1.4.2 Pre-production

This is in which each project begins. Essentially, pre-production defines what the game is about, why it needs to be made, and what it's going to take to make it.

You would possibly have the first-rate concept for a kind of game, a tale you need to carry to life, or you could need to construct one which leverages a positive kind of technology (e.g. VR, a brand new controller, or console). This stage can take a week to a year, relying on the challenge type, resources, and budget available, and normally takes as much as 20% of the overall manufacturing time. At this point, the group is pretty small. There can be a manufacturer, programmers, an idea artist (or if you're a one-individual operation, you'll be doing most of it!)[11].

the video game producer manufacturer handles the commercial things, in particular the financials. They control the price range and use advertising techniques to promote the product. A concept artist units the tone for the challenge early on through growing paintings and sketches. These early visuals assist shape the language of the sport, giving all of us operating at the challenge a visible manual to the general appearance and feel.

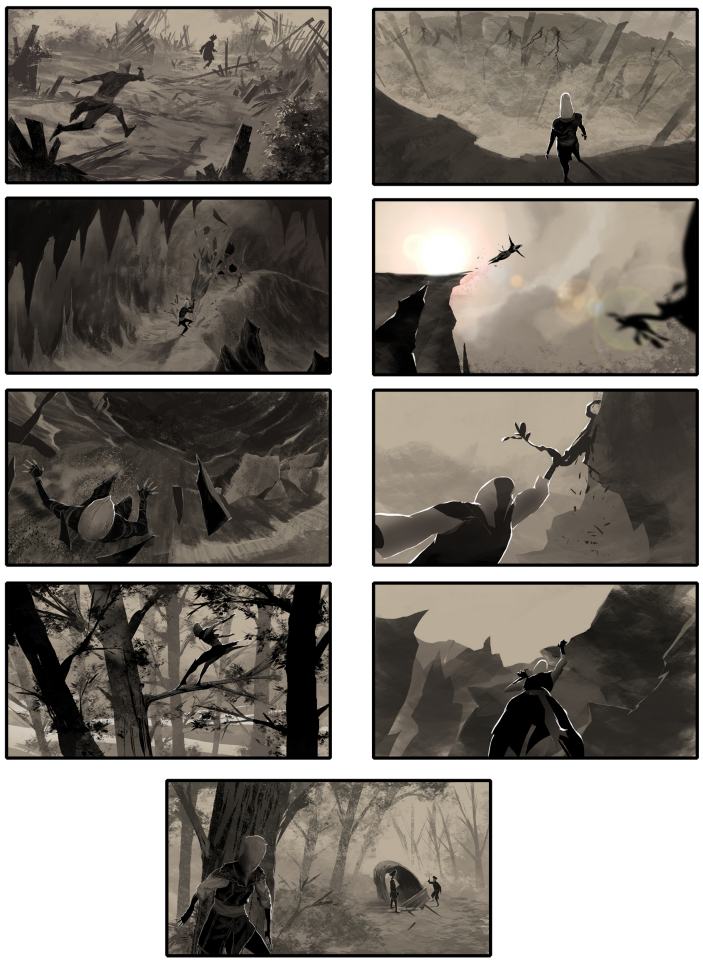


Figure 1.3: Assassin's Creed IV Black Flag Concept Art by Ivan Koritarev

.

A concept artist sets the tone for the project early on by developing artwork and sketches. These early visuals help highlight the features of the game, this giving everyone working on the project a visual guide to the general look and feel.

The information collected in this stage of pre-production forms the basis of the Game Design Document (GDD)[11].

1.4.2.1 The game design document (GDD)

The creation of the game design document is an essential step in the pre-production phase of the game, withinside the pre-production segment of the game, being answerable for guiding the project's scope (and as a result the complete production and improvement and testing of the game). A bad game design document may also end result in characteristic creep, and as a consequence, delays and losing milestones may occur.

Although there may be no general manner to construct build a game design document, Schuytema [Schuytema 2008] states that this report must have a complete description of the game in all its aspects, so that the development group can construct it. While describing the items, and characters in the game, it has to be documented not only what they do, but additionally what they affect, how they have interacted, and their function and conduct in the game. Despite the efforts to make the report pretty complete, the

report nevertheless might also additionally change. However, one has to examine the dangers of adjustments and if the time limits can nevertheless be met. the GDD includes such things as:
- The concept or idea.
- Genre.
- Story and characters.
- Core game mechanics.
- Gameplay.
- Level and world design.
- Art and/or sketches.
- Monetization strategy.

This document is later translated to a Product Backlog within the production phase, for little games it should be optional, translating the wants directly as a Product Backlog. this could save time for the team as they are going quickly to the production, however might also increase risks of feature creep or not a really entertaining game.

A GDD keeps the development team work organized, helps determine potential risks, and lets you see ahead of time who you may need to hire/outsource to so as to bring your project to life. Your game idea could seem fairly straightforward, however, once you lay it get into a GDD, you'll notice how big and resource-heavy your project is. Another reason to have a GDD is .

It is to help promote and finance the game. Potential investors will want to see a solid plan before investing. Finally, GDD will help market the product once it is ready for release[11].

1.4.2.2 Prototyping

A game prototype is a raw test that checks the functionality, user experience, gameplay mechanics, and art direction.

Prototyping happens in pre-production to check whether or not or not the game plan can work, and if it's worthy to pursue. several ideas don't create it past this stage. The team will usually begin with paper styles to test theories and total many of the nuances of a game or a series of systems quickly, simply, and value effectively. The aim is to make a prototype up and running ASAP to test if the ideas work and if the game is as fun as you had hoped. Prototyping may also reveal surprising challenges, that may doubtless change the whole course of the project. It's necessary to possess others to check your prototype too, as a result of things that are obvious to you, might not be to others.

Placeholder assets are wont to save time and money. These low-quality assets substitute for things like weapons and props throughout the first testing phase, and if approved, they're replaced with final, high-quality versions later on[11].Placeholder assets will be purchased or found at no cost online at intervals in game development software. They're usually pretty basic shapes, however, may also be a touch a lot of advanced.

(a) Asset Collection (b) Environment

Figure 1.4: Free assets with full environment project in unreal engine 4

1.4.3 Production

Once the pre-production section is complete and the general game project was finalized, the development enters the game production phase, and a currently larger of producers, designers, artists, and programmers are usually brought into the mix. Production is the longest stage of the process, Ranging anywhere from 1 to 5 years or more, production is where the game starts to take shape, The story is refined, assets (characters, creatures, props, and environments) are created, the foundations of play are set, levels and worlds are built, code is written, and more. but initial ideas don't continually translate thus well in reality, so because the job is done, the game is continually tested and perfected. Since gamers expect game graphics to be movie-quality, 75-90% of a game's budget goes to artwork. Production milestones There are a variety of milestones to complete the game development process.

1.4.3.1 Production milestones

There are a variety of milestones to complete the game development process.

Prototype : This is the initial take a look at the game (which happens in pre-production and is described thoroughly above), Some games can never go beyond this phase.

First playable : The first testable version, whereas it's still aloof from the final product, placeholders are replaced with higher-quality assets and the art design is added.

Vertical slice : A vertical slice is a fully playable sample demo that can be accustomed to pitching your game to studios or investors. starting from simply some minutes up to an hour, a vertical slice provides first-hand expertise of the game.

Pre-alpha : Most content is developed in the Pre-alpha phase. At this point of points in the development of the game, some important decisions must be made. Content may be cut or new items will need to be added to improve gameplay.

Alpha : The definition of the alpha stage varies from developer to developer; however, in general, it always means the game's "functionality" is essentially complete. This usually interprets to all or any of the specified options of the look having been implemented, however not all essentially operating specifically within the desired manner. From the beginning of development to succeed in the alpha stage is the longest part of the event cycle. this can be usually wherever some slippage can occur as features take longer to implement than planned, or the chosen technology either doesn't deliver the mandatory needs or demands additional work to bring to the expected state.

Beta : The beta stage effectively happens once all of the options that were delivered throughout the alpha stage are currently operating and bolted down so no core practicality at this stage is changed. Since all of the planned gameplay mechanics and technological features are implemented, the testing department is essentially disbursement its time making an attempt to "break" the game and providing minor gameplay tweaking feedback on areas comparable to issue settings, scoring, or points systems, then forth. throughout the beta, the ultimate sound effects, musical score, voice talent, and localization are all extra and completed, with constant testing and feedback from QA to make sure that every one content is up to the required customary and within the right places[11].

These tasks are typically handled throughout beta merely thanks to the developers having the ability to focus a lot on content-driven problems over the technical tasks that dominate alpha. throughout the beta, the press and public are going to be given more data regarding the game in the kind of in-depth "hands-on" features, and because the game nears completion, preview reports. For laptop titles that include an internet part (and even for a few console games), it's turning into more and more common for developers to unharness reduced content "beta test" builds of their game for controlled public testing. This facilitates nailing down hardware compatibility problems distinctive to laptop titles and provides the event team real shopper feedback to supplement that provided by QA.

Public beta tests can even help developers see their content during a real-world setting that's not possible to recreate altogether however the biggest company QA departments. Factors comparable to stress testing are often viewed with the acceptable action to repair issues or issues before release. it's also throughout beta that games are below the most important threat of being leaked lawlessly to the public.[9] Usually, this happens through an enclosed company leak (e.g., by unscrupulous freelance QA testers) or through scalawag journalists and generally even hardware partners. In most cases, developers and publishers have internal procedures to assist try and forestall this, comparable to digital watermarking, though there are still varied unauthorized releases of beta code each quarter, typically on the best profile titles.[12]

Gold master : The gold master is named as when the gold-colored recordable discs were originally used to send final mastering assets to the publisher for mass duplication. it's at this final development stage for computer titles that replicate protection is another, installation computer code is integrated, and device drivers are added to the game.

Once all of the essential bugs are far away from the game and everyone party agrees that the necessities of the beta definition have been met, the game is asserted a gold

candidate and is shipped for duplication. it's sometimes at now that work is finalized on a particular demo for magazine cowl mounts and online distribution, though more and more groups are building specific demo content and previews into the initial schedule and value breakdown, particularly within the case of console developers, thanks to the lead times on magazines.

1.4.3.2 Key game development roles

Game dev roles can vary reckoning on the size and sort of studio. These are a number of the common positions you'll find.

Project manager : The project manager makes certain the game development process runs smoothly, milestones are met, risks are anticipated/mitigated, and team members do what they're supposed to. they're usually the center of communication between the dev and design teams and executives. Project managers are exceptionally organized and should have glorious communication skills.

Game developers / programmers : Game software engineers facilitate developing games by turning style ideas into code to make playable games. Programmers are usually software system engineers or laptop scientists with a robust programming background, and a mixture of creativity, mathematics skills, and patience to success code ideas into interactive visuals and sounds. They make sure the game runs smoothly. There are many different aspects of programming, including:
- Building a custom base engine for the game (optional).
- Scripting functions, events, interactions.
- making physics (e.g. gravity variations during a game set in space).
- Developing and creating 3D graphic renders.
- Simulating AI in opponents.
- Adding sound effects, and voice-overs.
- Implementing game logic and mechanics.
- making the program.
- Writing code specific to joysticks, keyboard, mice.
- creating it attainable for players to contend or work via local area networks or the net.
- Developing custom tools. Cross-platform porting code.
- Implementing algorithms, addressing memory needs, and caching issues.
- Identify and fix bugs.

In larger studios, we'll find specialists dedicated just to AI programming for the game, or programmers who solely work on the user interface. the game programmer (simple programmer) makes bout USD 60,000 per year[11], however, a senior or lead programmer will earn far more than USD 100,000 per year.

Game designers : A game designer is the creative engine of the game, and usually a cross between a author with artist, with information of programming. the game design production section involves making compelling stories, characters, goals, rules, and challenges that drive interactions with different characters, users, or objects. Designers is also

accountable for:
- Developing the storyline, character back-stories, and dialogue.
- Developing gameplay, rules, and also the classification system.
- Determining the level of difficulty.
- Building environments, ledges, obstacles, and objects.
- Level and world design.
- Programming/scripting.
- Digital written material.

the common game designer pay is USD 64,000, whereas a lead will build in more than USD 94,000 (depending on experience, location, size of studio, and industry). Senior and technical designers could exceed this range.

Level designers : A video game level designer is responsible for creating fun and interesting levels. Their job is to keep the player focused on moving through the game and achieving their goal or mission Clearly and simply while minimizing the potential for confusion.

With games becoming a lot more complex than they used to be, it's common for large studios to find game designers dedicated only to the design level. Depending on the type of game and whether it's based on actual events (such as a World War II battle), they may need to learn all about a certain period of history and research actual locations to make sure the level is realistic and believable. If the game is adapted from a book or movie, they will need to read/watch the original and look for clues. If the world is completely fictional, they will need to use their creativity and draw inspiration from the concept of art presented.

Then using a level editor (software used to design levels and maps), they build levels, stages, or missions. Their job can also include things like planning start and exit locations, locating hidden tunnels and passages, places where interactions or dialogue will occur, monster spawn points, starting points where certain actions will occur, and much more.

Level designers are responsible for identifying and fixing errors, such as players falling out of bounds or getting stuck and unable to get out. The challenge with video games is that once the game is released, it is out of the designer's hands. The player can interact with the world in unexpected ways, highlighting errors that went unnoticed during development.

The level will likely see several versions before it becomes final.

A level designer can earn an average of USD 57,000 depending on the studio and location. Experienced or experienced designers can command a much higher salary.

Game artists : Game artists will include concept artists, animators, 3D modelers, and FX artists. This cluster is answerable for transferring color, movement, and life to the game. whereas the concept artist is usually active throughout pre-production once coming up with the initial look (usually in 2D), it's going to be brought back later within the game development method if new components are added or the course of the game is changed. A 3D concept artist (which can be a similar artist) uses digital sculpting and

texturing software like Blender, ZBrush, Maya, Quixel Mixer , Gimp, and Photoshop to form 3D props, assets, and environments[11].

3D modelers : A professional 3D creator could be a sculptor or a technician. he's a creative person and an engineer. He should worry about the form, expressiveness, and style, as well the topology, and potency of vertex count. though there are several ways and kinds of modeling, in play nowadays polygon modeling is king[1].

animators : An animator within the games business is predominately a personality animator since moving and talking characters are normally required in most game genres. The animation of the many different styles of characters is a kind that takes a lot of time and follows to perfect, therefore creating character animation a lot of specialization.[8] As competent game animators, it's their job to grasp the means behind an expression and the way to get the body displayed and moving to accurately or stylistically convey the action for playback in real-time. Associate in Nursing animator ought to even be knowledgeable in anatomy, since making likely motion requires Associate in the Nursing understanding of the underlying mechanism, like rotating joints and bulging muscles. Character work may be a large order and usually requires seasoned animators. Moreover, because the spoken language goes, you are doing get what you obtain once a game company is in search of tough talent[3].

visual effects artists (FX): The role of an FX effects artist is arguably one of the foremost difficult of the art jobs in games. It demands technical savvy, a way of animation and timing, a grasp of physics, a deep understanding of the game engine, additionally the ability to color textures of natural phenomena. Effects also need a general understanding of adjacent disciplines that you just can act with, as well as scripting, code, animation, prop modeling, and rigging. the results artist will bring a game engine and frame rate to its knees quicker than anyone else by filling the screen with smoke. FX effects can also be one of the most rewarding art jobs[1].

Audio engineers / sounddesigners / composers : Modern game projects need a whole crew of audio professionals to induce the duty done well. Music supervisors, sound designers, audio directors, voice actors, implementers, audio software package engineers, casting agents, musical organization contractors, dedicated music executives, and more. manufacturing audio for games takes ability and data of music and sound so theand also the tools concerned in making those elements.[8] making and producing sensible music could be a craft that takes years to hone. an equivalent are often aforementioned for excellent sound design. The role of the producer is to ensure the simplest talent is functioning on the project—and therefore he ought to have level of familiarity with what's concerned in game audio production[5].

QA (quality assurance) / video game testers : The role and methodology of QA checking vary from company to company; most developers don't have full-time "in-house" QA departments and think about numerous self-moderated ways in which of

bug-fixing their code, typically going most of the deficiency or noncritical minor bugs to the QA departments of their publisher to select up throughout the beta stage. throughout the beta stage and thru to the completion of the project, the game will be under constant test from the publisher's QA department (or increasingly by dedicated "professional" test teams at testing service outsourcing companies). Most feedback given to the team can come back in the sort of gameplay flaws, issue issues, and technical bugs about graphics, sound, or hardware incompatibility, the latter of that being notably necessary for computer product[3].

For developers of console products admire the Microsoft Xbox and Sony PlayStation systems, there'll be an extra layer of QA at the platform holder' location (sometimes in many totally different countries) wherever further guideline or technical demand list (TRC) testing is applied to confirm the game adheres to the varied internal control standards that platform holder has place into place. The testing procedures for console products are abundant tighter than developing for the other platform, since mending is harder thanks to restricted on-line connectivity, whereas computer developers are often forced to go away minor bugs to repair till when launch, safe within the data that they'll forever patch the matter later.[8] this is often often through with an extra transfer from a support web site or specialist online game sites and magazine cowl disks. The timeline for platform TRC testing varies slightly, however Sony final approval (or submission as it' referred to) takes between 4 to 6 weeks to complete, with a further 2 to 4 weeks supplementary if the game "fails" submission and desires to be fastened/sent back. The producer and also the publisher' external producer will act because the buffer between QA and the development team to confirm they're not flooded with haphazard bug reports. Most corporations use bug-tracking programs, machine-driven spreadsheets, or Oracle info systems to manage and range bug feedback from QA. Key team members will have periodic bug update conferences to ensure that fixed bugs are being off from the project and the reports from QA are that specialize in the forthwith necessary areas.[1]

1.4.4 Post-Production

The final stage of a game's development is the post-production stage. This begins once the game is taken into account "feature complete" and every one of the codes has been written associated art has been completed. this can be when an alpha version of the game is made and is provided to the game' take a look at the department to bang away at and notice bugs and major flaws within the game that requires to be modified whether or not by the artists or programmers.

Once all of the bugs and major flaws are known and addressed, a beta version of the game is then created and another time sent to take a look at the department to be picked through with a fine-tooth comb. this can be wherever the hardcore testing is completed and every single bug notwithstanding however major or minor is documented and tried to be fixed, with the most important "A" sort bugs the highest priority with the "B"', "C"" and fewer important bugs self-addressed as time or company policy might once developing a title for any of the consoles by corporations comparable to Microsoft, Nintendo or Sony, this is additionally the stage where the testers should confirm that the game abides by all

of the "standards" that are determined by the manufacturer of the console that has got to be followed so as for the game to be approved for release. Includes elements comparable to the "B" button should continue to be used menusgames developed for the Microsoft Xbox and therefore the "A" button always has to be used to advance.

Once all of the bugs are mounted and every one of the standards has been determined to be met, a final version of the game is created and, within the case of the consoles, is distributed to the console maker to urge take a look at and approved for unharness on the system in If bugs are found or approval isn't met, the assembly team can fix all of the issues in question, place it through their test department again to make sure that everything was mounted and zip new was broken, and so another time submit it for approval.

All that's left to try and do once the game is approved by the console manufacturer or simply "finished" by the developer within the case of laptop games, is for the game to be factory-made and so distributed to stores wherever you'll be able to quit and get them[4].

1.5 Conclusion

Based on what we see later, there are multiple phases within a game production, each with its own goals and set of objectives. Additionally, it was investigated what each phase would be focused on and when each phase would be considered complete. This led to the indication that the most time-consuming and thus the most uncertain phase would be the production and the team size could potentially vary greatly depending on the game concept as well, although the team size or economical limitations could, in reverse, help shape the game concept. Moreover, some team members could potentially be part of two departments simultaneously and need to be updated on both while still maintaining a clear overview.

Having gotten a better understanding of the production phases, the next point of interest would be to focus on the game development process and looking into work engagement there were several types of management methodologies. However, not all would be equally fit for running a game production as a game production tends to involve a lot of changing requirements.

Chapter 2

Artificial intelligence in Game development

2.1 Introduction

It may seem unexpected, but the history of AI began long before computers. Even the ancient Greeks suspected the existence of intelligence machines. A famous example is the bronze giant Talos, who protected the city of Crete from invaders. In , René Descartes wrote about thinking automata, believing animals were unlike machines that could replicate themselves using rollers, pistons, and cams. However, the heart of this story begins in 1931, when the Austrian logician, mathematician, and philosopher Kurt Gödel showed that all true statements in first-order logic are differentiable. On the other hand, this is not true for higher-order logic. , in which some true (or false) statements are not provable. This made first-order logic a good candidate for automating derived logical consequences.

In 1937 Alan Turing, an English computer scientist, mathematician, logician, cryptanalyst, philosopher, and theoretical biologist, pointed out some of the limitations of "intelligent machines" with the stopping problem: It is not possible to predict a priori whether a program will stop, when not running. This has many consequences in theoretical computer science. However, the fundamental step came thirteen years later, in 1950, when Alan Turing wrote his famous article " Computing Machinery and Intelligence ", in which he talked about the imitation game better known today as " The Turing Test ": a way to the definition of what an intelligent machine is. From the late 1950s to the early 1980s, much AI research was devoted to "symbol systems". These are based on two components: a knowledge base made up of symbols and a reasoning algorithm that uses reasoning to manipulate these symbols to extend the knowledge base itself.

During this period, many brilliant minds made significant advances. A name worth mentioning is McCarthy, who organized a conference at Dartmouth College in 1956 where the term "artificial intelligence" was first coined. Two years later he invented the high-level programming language LISP, in which the first self-modifying programs were written. Other notable results include Gelernter's Geometry Theorem Prover from 1959, Newell and Simon's General Problem Solver (GPS) from 1961, and Weizenbaum's famous Eliza chatbot, which became the first software to have a natural language conversation in 1966.

The apotheosis of symbolic systems finally came in 1972 with the invention of PROLOG by the French scientist Alain Colmerauer.

Symbolic Systems gave rise to many AI techniques that are still used in games, such as B. Whiteboard architectures, pathfinding, decision trees, state machines, and direction algorithms. The compromise of these systems is between knowledge and search. The more knowledge you have, the less you need to search, and the faster you can search, the less knowledge you need. This was even proved mathematically by Wolpert and Macready in 1997. In the early 1990s, token systems became inadequate as they proved difficult to scale to larger problems. Additionally, some philosophical arguments have been advanced against it, arguing that symbolic systems are an incompatible model for organic intelligence. As a result, old and new technologies inspired by biology have been developed. With the success of Nettalk in 1986, a program for reading aloud, and the publication of the book Parallel Distributed Processing by Rumelhart and McClelland that same year, the old neural networks were swept off the shelf. In fact, "back-propagation" algorithms have been rediscovered as they allow a neural network (NN) to learn.

in terms of games, the quality of the experience depends on whether the game presents a good challenge to the player. One way to present a good challenge is to offer computer opponents, or sometimes even allies, that are capable of playing the game intelligently. In most cases, this is not a trivial problem to solve, but fortunately, there is an entire field of study that can help us ou—artificial intelligence (or AI for short). AI describes the intelligence embodied in any manufactured device. If we design a character or opponent in a video game that acts on its own, it is generally accredited with possessing AI. Human-level AI is the stuff of dreams and science fiction. How do you take the accumulated common sense and expertise of a human and distill it into a computer? Unfortunately, this problem is currently unsolved, and it will likely be decades before we get a close understanding of what it truly entails. Since general human-level intelligence is currently impossible to re-create, researchers chip away from dozens of different angles by solving much simpler problems. By sufficiently narrowing down the domain of an AI problem, it becomes possible to create behavior that is reasonable and believable, especially in the realm of video games[1].

2.2 AI In Games

AI in video games is very different from most other AI applications like military defense, robotics, or data mining. The main distinction concerns the goals. The goal of an AI programmer is to create fun and challenging opponents while the product is being shipped. in time. These goals have the following five implications:

The AI must be intelligent but intentionally flawed :
- Opponents must be challenging.
- Opponents must keep the game fun and entertaining.
- The opponents must lose to the player in a challenging and fun way.

The AI must not have any undesirable weaknesses :
- There shouldn't be "golden paths" to beating the same way every time.
- AI shouldn't fail miserably or look stupid.

The AI must function within the game's CPU and memory limitations :
- Most games are real-time games and their AIs need to react in real-time.
- Game AI rarely gets more than 10-20 percent of the frame time.

The AI should be configurable by game designers/players :
- Designers need to be able to adjust the difficulty, customize AI, and sometimes script specific interactions.
- If the game is expandable, players can modify or customize the AI.

The AI must not stop broadcasting the game :
- Experimental techniques should be tested early in the development cycle during pre-production.
- The AI techniques used must not endanger the game.
- If AI is given the freedom to evolve or change, it must be testable to ensure it will not degrade when released to millions of consumers[1].

These requirements shape the perception of game developers in the field of AI. An important difference is that the game's AI doesn't have to solve a problem perfectly, only to the satisfaction of the player. For example, pathfinding may require the AI to calculate a path through a room full of people. There are search algorithms to find the shortest or cheapest route, but perfection is not generally a requirement for games. By relaxing the standards for many problems, shortcut decisions can be made that either make the problem manageable in real-time or result in large computational savings.

Another consequence of game-specific AI is that the AI has access to perfect knowledge. For example, a given opponent doesn't have to feel the world the way a physical robot would. The game world is entirely in the computer and the AI has the luxury of doing its analysis on these completely accurate representations. Much of robotics research focuses on problems of visual recognition and mechanical movement, both of which are rightly ignored in games. When designing game AI, game designers should think hard about making the AI configurable.

Rather than creating a perfect standalone character or opponent, the goal is to create a highly customizable AI that can be tweaked based on difficulty and individual attributes like aggressiveness or accuracy. [13]By creating a slightly more general AI that can be customized, game design experts can balance and tweak the game to ensure the game is fun. Finally, an important consideration is that the product must be shipped on time. Experimental AI techniques are exciting and intriguing, but they have the potential to unnecessarily jeopardize the project. Therefore, new AI techniques should be tested early in the development cycle. I promise everything being assembled three months before shipping is just not acceptable.

2.3 Game Agents

In most games, the goal of the AI is to create an intelligent agent, sometimes called a non-player character (NPC). This agent acts as an opponent, ally, or neutral entity in the game world. Since most game AI focuses on the agent, it is very useful to examine game AI from this perspective. An agent has three key steps that it continuously goes through.The stages are commonly referred to as the sensory-thinking-acting cycle. In addition to these three steps, there is an optional learning or remembering step that can also take place during this loop. In practice, most game agents don't take that extra step, but that's slowly changing due to the added challenge and replayability that comes with it[13].

2.3.1 Sensory

2.3.1.1 Sensing

The game agent must have information about the current state of the world to make good decisions and act accordingly. Since the game world is presented entirely within the program, perfect information about the state of the world is available at all times. This means that there is no uncertainty about the world. The world provides the game agent with accurate information about the existence, location, and status of any opponent, barrier, or object[13] .

Unfortunately, while all this rich information is available, it can be expensive or difficult. for useful and relevant information. The game agent can always consult the game world representation to locate the player or other enemies, but most players would consider this cheating. Therefore, it is necessary to give the game's agent certain limitations in terms of what he can feel. For example, it may seem obvious, but game agents shouldn't be able to see through walls in general. Gaming agents generally have human limitations. They are limited to only knowing about events or entities that they have seen, heard, or perhaps told them. by other agents. Therefore, there is a need to model how an agent should be able to see the world, hear the world, and communicate with other agents[23].

2.3.1.2 Vision

When modeling agent vision, it is important that the game engine provide fast methods for determining the visibility of objects. While game AI typically isn't very CPU intensive, visibility testing can be enormously expensive. Therefore, it is often limited to particular agents and performed only periodically[13].

Vision usually starts with obtaining a list of pertinent game objects. For example, the agent might ask for a list of all enemies. Since agents are not concerned with most game objects that populate a world, it would be wasteful to consider every object in the game database. Once this pared-down list is constructed, a vector from the game agent to each game object is calculated.[25] This object vector is then processed in the following ways to determine if the agent can see the game object. The order of these steps is important to minimize processing.

2.3.1.3 Hearing

An interesting twist on agent awareness is allowing an agent to sense through hearing. For example, if the player tiptoes past a sleeping enemy, the enemy may not notice. However, if the player walks past the same enemy, the enemy can hear the sound. and wake up. If the player starts shooting their gun wildly, agents who cannot see the player may rush to the crime scene because they heard gunfire from that location[13].

Hearing is most commonly modeled through event-based notifications. For example, if the player performs an action that causes noise, the game calculates where that noise might travel and informs the agents within that area. Instead of carrying out complex calculations of the sound reflection about the environment, this is usually achieved by simply calculating the distance together with the delimited areas. If a noise emanates within Area B and can be heard up to 10 meters away, all agents within Area B and 10 meters will be notified. This eliminates any computationally intensive sound modeling.[14]

2.3.2 Thinking

Once an agent has gathered information about the world through their senses, the information can be evaluated and a decision made. This mindset is at the core of what most people consider real AI, and it can be as simple or as sophisticated as needed. In general, there are two ways an agent decides games. The first is that the agent relies on pre-coded expertise, typically built via if-then rules, introducing randomness to make agents less predictable.[13] The second is that the agent uses a search algorithm to find a near-optimal solution.

2.3.3 Acting

Until now, the game agent's perception and thought processes were invisible to the player. Only in the action step can the player witness the intelligence of the agent. Hence, this is a very important step for the agent to execute the chosen decisions. and communicate their decisions to the player (if this improves the game and the player's perception of the agent). In other words, if the agent is brilliant and the player never notices, the effort of making the agent smart was wasted.[13] There are numerous agent actions in the game. Some common ones are changing location, playing an animation, playing a sound effect, picking up an item, talking to the player, and firing a weapon.

The skill and subtlety with which the agent performs these actions affect the player's opinion of the agent's intelligence. This places a tremendous strain on the variety and aesthetic quality of the animations, sound effects, and dialogue created for the agent. The agent can only express their intelligence in terms of the vocabulary these artistic resources provide. On matchdays, agents had very little animation to deal with. As 3D games appeared, the agent's repertoire expanded from several dozen animations to hundreds and thousands. This complexity led to the need to handle animation selection in a scalable way. Best practices in this area have brought the problem of animation selection through data-driven design out of the code and right into the hands of game designers and creators. It is important to pass the hidden work on to the player as it

improves the game. For example, if the agent has concluded that he will inevitably die soon, there may be nothing he can do to prevent that outcome. However, when the agent just sits and dies, it looks pretty silly.

The result would be that the agent would use this information to pinch or yell "Oh no!" when he is dying. This way players don't see a dumb agent getting killed; Instead, they see an intelligent agent who understands the situation. Although the result is the same, revealing intelligence greatly improves the agent and the game[14].

2.3.3.1 Communication

Many types of agents are expected to communicate with each other, so it can be important to model the transfer of perceived knowledge between agents. Take the guards for example. If a guard saw the player in a sensitive area, they could run away.[13] and warn others. The other guards could use this information to make better decisions themselves, such as deciding to hunt the player together, starting with the player's last known location. Similar to the mechanism of hearing, communication information will be event-based. the form of notifications. When an agent has useful information and gets within a certain distance of other agents, the information is sent directly to the other agents. [49]

2.3.3.2 Reaction Times

When capturing the environment, it is important to build in artificial reaction times. Agents should not be able to see, hear, or communicate immediately. For example, seeing a guard take off at the exact moment the alarm sounds seems decidedly wrong. is made to sound Because agents perceive the world instantly, simple timers can be used to simulate reaction times. Typical reaction times for seeing and hearing can be on the order of a quarter to half a second.Communication reaction times would be longer to model speaking or gesturing between agents.

2.3.3.3 Search

Search is another technique commonly used to make smart decisions. The search uses a search algorithm to discover a sequence of steps (a plan) that leads to a solution or ideal state. Given the possible moves and the rules governing the moves, an algorithm can explore the search space and find an optimal or near-optimal solution if one exists. In games, fetch is most commonly used to plan where the agent should move next in many games[13].

2.3.4 Learning and Remembering

learning and remembering together is an optional step in the Sensing-thinking-acting cycle. Without it, the agent will never improve, never adapt to a specific player, and never benefit from past events or information witnessed or shared with him. Interestingly, learning and remembering aren't necessarily important in many games, simply because agents may not live long enough to benefit from what they've learned.[13]

However, in games where the agent lasts longer than 30 seconds, there can be a significant benefit when learning and retrieval are built-in. For game agents, learning is the process of remembering specific outcomes and using them to generalize and predict future outcomes. Typically this can be modeled using a statistical approach. By collecting statistics about past events or outcomes, future decisions can use these probabilities. For example, if the player is attacking from the left 80 percent of the time, the AI would be smart to wait and prepare for that likely event Therefore, the AI was adapted to the behavior of the player.[38]

Remembering can be as simple as noting the last place the player was seen with that information during the thought cycle. By retaining some accounting information about observed states, objects, or players, the agent can exploit earlier observations at a later time. In order not to accumulate too much knowledge, these memories can fade over time depending on how important they are. be a way to model selective memory and forgetting. It is important to note that past knowledge does not always have to be stored in the agent. Some types of knowledge can be stored directly in the world's data structures. (This refers to smart terrain) For example, if agents are constantly being massacred in a certain location, that area can be marked as more dangerous. You could think of it almost like the smell of death in a certain place. During the thought cycle, path planning and tactical decisions may take this information into account and prefer to avoid the area.[14]

2.4 Multi-Agent system

A multi-agent system (MAS) is a computerized system composed of multiple interacting intelligent agents.[15] Multi-agent systems can solve problems that are difficult or impossible for an individual agent or a monolithic system to solve.[16] Intelligence may include methodic, functional, procedural approaches, algorithmic search, or reinforcement learning.[17][18] To create a multi-agent model, the smaller components that comprise the system must be specified [19], These smaller components need to be fully modeled, to become the agents that are at the heart of the modeling technique [19].

Each agent must be capable of making decisions (often dictated by a rule-set), and these decisions may involve incomplete knowledge of its environment , The agents also need to be able to receive information from their environments and depending on the type of system being modeled, sometimes communicate with other agents , The environment itself needs to be able to adjudicate the interactions between agents, but at no stage needs to be able to determine the overall ramifications of these interactions. Instead, the overall result will become apparent empirically, taking advantage of the emergent behavior of the MSA to handle the complexity modeling [23].

The field of MAS is a well-established research domain in AI, which has an emphasis on the resolution of problems by a society of agents. The distribution in several agents is necessary because these problems can be complex or too large to be solved by a single process, or even, they can need knowledge of several domains. MAS researchers look for 'autonomy' because the more autonomous the system is, the more efficient the task distribution and execution, and the lower the computational load of the overall system.

However, a MAS comprises the formalization of the coordination, hierarchical relationships, and communication between agents. The MAS field is increasingly characterized by the study, design, and implementation of societies of artificial agents. If the logic-based and cognitive science approaches have contributed considerably to the development of MAS, the inverse does not happen (the social sciences have been less influenced) [22]. Just in the economics and game theory area, there is a huge quantity of work in the M AS field [21]. In these areas, MAS is essentially used by economists and game theorists to study the evolution of cooperation from local interactions among self-interested agents.

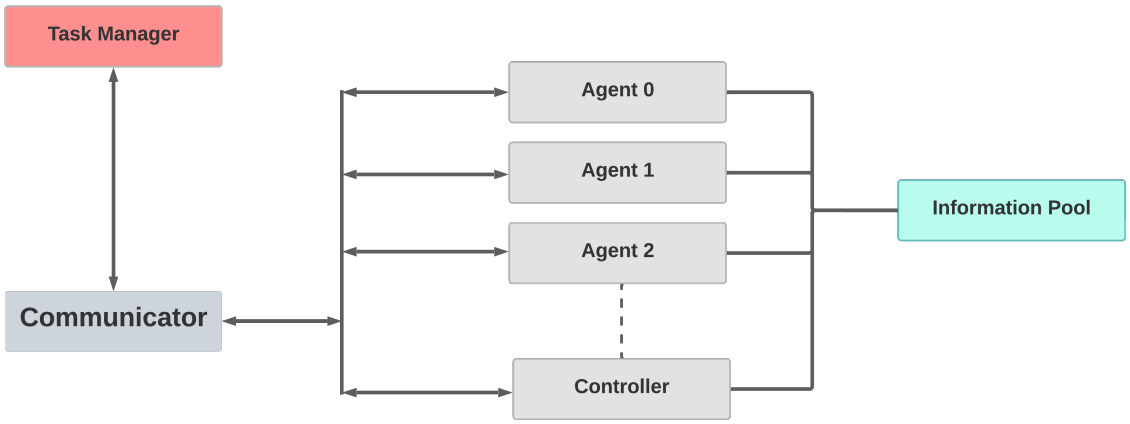


Figure 2.1: An example of a Multi-Agent System for decision support[49]

2.5 Behavior Tree

There are several abstractions to help design complex behaviors for an autonomous agent. Generally, these consist of a finite set of entities that map to particular behaviors or operating modes within our system, e.g., “move forward”, “close gripper”, “blink the warning lights”, “go to the charging station”. Each model class has some set of rules that describe when an agent should execute each of these behaviors, and more importantly how the agent should switch between them[34].

Behavior trees (BTs) are one such abstraction, which we will define by the following characteristics:

They start at a root node and are designed to be traversed in a specific order until a terminal state is reached (success or failure). Leaf nodes are executable behaviors: Each leaf will do something, whether it’s a simple check or a complex action, and will output a status (success, failure, or running). In other words, leaf nodes are where you connect a BT to the lower-level code for your specific application. Internal nodes control tree traversal: The internal (non-leaf) nodes of the tree will accept the resulting status of their children and apply their own rules to dictate which node should be expanded next[35]. Behavior trees actually began in the videogame industry to define behaviors for NPCs :

Both Unreal Engine and Unity (two major forces in this space) have dedicated tools for authoring BTs. This is no surprise, a big advantage of BTs is that they are easy to compose

and modify, even at runtime. However, this sacrifices the ease of designing reactive behaviors (for example, mode switches) compared to some of the other abstractions.

Since then, BTs have also made it into the robotics domain as robots have become increasingly capable of doing more than simple repetitive tasks.

2.5.1 Behavior tree terminology

While the language is not standard across the literature and various software libraries, We will largely follow the definitions in Behavior Trees in Robotics and AI. At a glance, these are the types of nodes that make up behavior trees and how they are represented graphically[35]:

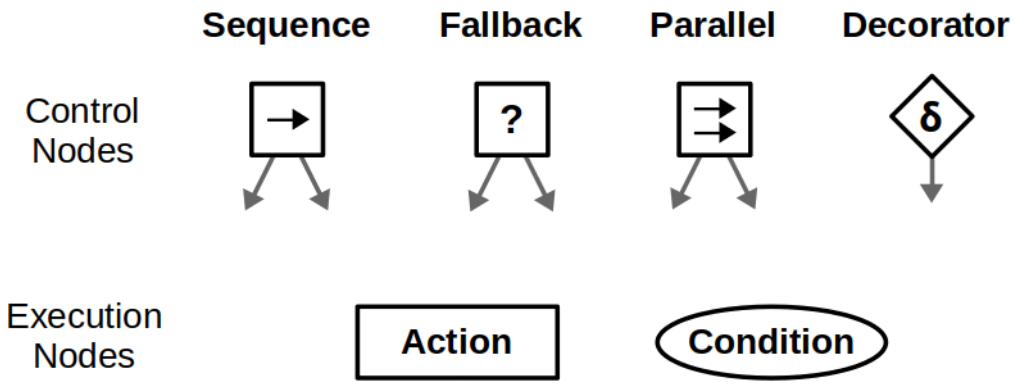


Figure 2.2: Overview of behavior tree nodes.

Behavior trees execute in discrete update steps known as ticks. When a BT is ticked, usually at some specified rate, its child nodes recursively tick based on how the tree is constructed. After a node ticks, it returns a status to its parent, which can be Success, Failure, or Running[47].

Execution nodes, which are leaves of the BT, can either be Action or Condition nodes. The only difference is that condition nodes can only return Success or Failure within a single tick, whereas action nodes can span multiple ticks and can return Running until they reach a terminal state. Generally, condition nodes represent simple checks (e.g., “is the gripper open?”) while action nodes represent complex actions (e.g., “open the door”).[35]

Control nodes are internal nodes and define how to traverse the BT given the status of their children. Importantly, children of control nodes can be execution nodes or control nodes themselves. Sequence, Fallback, and Parallel nodes can have any number of children, but differ in how they process said, children. Decorator nodes necessarily have one child and modify its behavior with some custom-defined policy.[34]

2.5.1.1 Sequence :

Sequence nodes execute children in order until one child returns Failure or all children returns Success.[35]

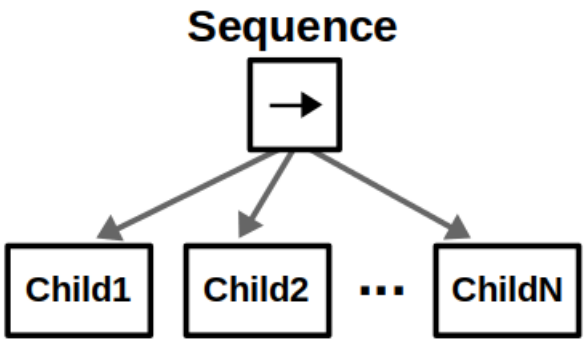


Figure 2.3: Sequence node.

2.5.1.2 Fallback :

Fallback nodes execute children in order until one of them returns Success or all children return Failure. These nodes are key in designing recovery behaviors for your autonomous agents[35].

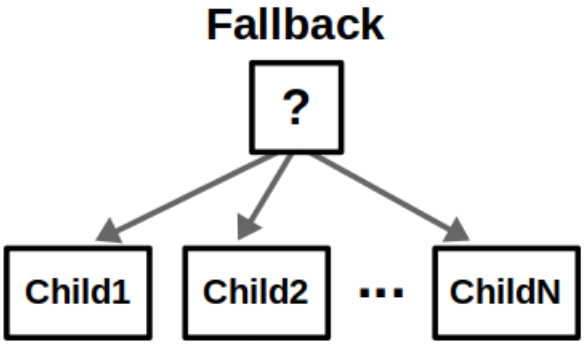


Figure 2.4: Fallback node.

2.5.1.3 Parallel :

Parallel nodes will execute all their children in "parallel". This is in quotes because it's not true parallelism; at each tick, each child node will individually tick in order. Parallel nodes return Success when at least M child nodes (between 1 and N) have succeeded, and Failure when all child nodes have failed.

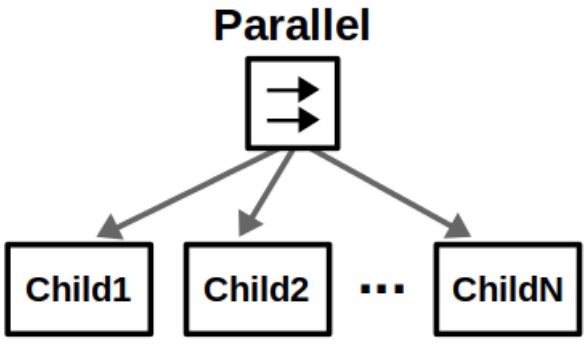


Figure 2.5: Parallel node.

2.5.1.4 Decorator :

Decorator nodes modify a single child node with a custom policy. A decorator has its own set of rules for changing the status of the "decorated node". For example, an "Invert" decorator will change Success to Failure, and vice-versa. While decorators can add flexibility to your behavior tree arsenal, you should stick to standard control nodes and common decorators as much as possible so others can easily understand your design.[35]

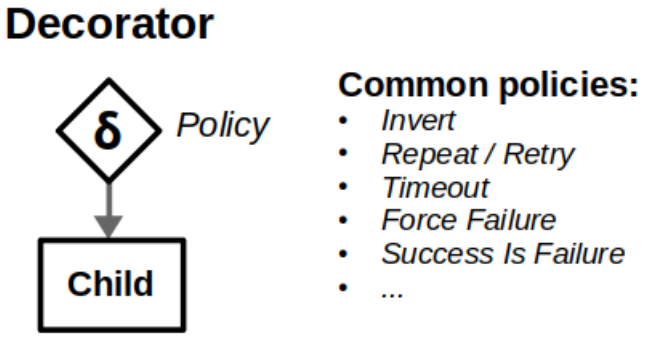


Figure 2.6: Decorator node.

2.5.1.5 Execution node :

Execution nodes are leaves of the behavior tree where tasks are executed.[47]

2.6 Common AI Techniques

The following survey of common AI techniques is designed to provide an executive summary of the many tools that an AI programmer can wield. Since game AI is approached from so many diverse directions, a whirlwind tour of techniques is a good way to familiarize oneself with the diverse landscape of available solutions. The next section similarly provides a survey of promising AI techniques[25].

2.6.1 Finite State Machines

Finite State Machines (FSMs) are generally recognized as the most widely used software pattern within game AI. This kind of popularity is no accident. Rather, FSMs are widely used because they have some amazing properties. They are easy to program. , easy to understand, easy to debug, and completely general for any problem. They may not always provide the best solution, but they always get the job done with minimal risk to the project. However, FSMs also have a darker side. Many programmers view them with suspicion as they tend to be created ad hoc with no consistent structure. They also tend to get out of control as the development cycle progresses. This poor structure coupled with growth makes many FSM implementations difficult to maintain and brittle. However, for all their warts, FSMs remain the most compelling way to structure most game AI implementations[20].

2.6.1.1 The basics of Finite State Machines

Formally, a finite state machine is an abstract computational model consisting of a set of states, an initial state, an input vocabulary, and a transition function that maps inputs and current states to the next state. [24]The calculation starts from the initial state and transitions to new states as inputs are received. The FSM can perform work within a particular state, known as a Moore machine, or at transitions between states, known as a Mealy machine. Game developers deviate from the strict definition of FSM in many ways. First, the states themselves are used to define behaviors that contain code specific to that state.[35]

For example, states can be behaviors such as walking, attacking, or fleeing. Second, the single transition function is usually distributed across the states so that each state knows exactly what will cause its transition to another state, which helps maintain the relationship between the states. and easy-to-understand transitions. Third, the boundary between Moore and Mealy machines is fluid, since work is often done both within a state and during transitions. Fourth, when transitioning into a new state, it is extremely common to exploit chance and randomness. For example, an agent might flee after being attacked with a 10 percent chance. Fifth, extra state information not directly represented in the FSM, such as agent health is often used as a deciding factor for some state transitions. Since FSMs can elegantly capture the mental states or behaviors of an agent, they are a natural choice for defining character AI[24].

2.6.2 Blackboard Architecture

A blackboard architecture is designed to solve a single complex problem by posting it on a shared communication space, called the blackboard. Expert objects then look at the blackboard and propose solutions. The solutions are given a relevance score, and the highest-scoring solution (or partial solution) is applied.[27] This continues until the problem is "solved."

Game Example: In games, the blackboard architecture can be expanded to facilitate cooperation among multiple agents. A problem, such as attacking a castle, can be posted, and individual units can propose their role in the attack.

The volunteers are then scored, and the most appropriate ones are selected[26]. Alternatively, the blackboard concept can be relaxed by using it strictly as a shared communication space, letting the individual agents regulate any cooperation. In this scheme, agents post their current activities and other agents can consult the blackboard to avoid beginning redundant work. For example, if an alarm is sounded in a building and enemies start rushing the player, it might be desirable for them to approach from different doors. Each enemy can post the door through which it will eventually enter, thus encouraging other enemies to choose alternate routes[27].

2.6.3 A* Pathfinding

A* pathfinding (pronounced A-star) is an algorithm for finding the cheapest path through an environment. Specifically, it is a directed search algorithm that exploits knowledge about the destination to guide the search intelligently. By doing so, the processing required to find a solution is minimized. Compared to other search algorithms,A* is the fastest at finding the absolute cheapest path[28][29][30]. Note that if all movement has the same traversal cost, the cheapest path is also the shortest path.

Game Example: The environment must first be represented by a data structure that defines where movement is allowed[41]. A path is requested by defining a start position and a goal position within that search space. When A* is run, it returns a list of points, like a trail of breadcrumbs , that defines the path. A character or vehicle can then use the points as guidelines to find its way to the goal. A* can be optimized for speed [28][29][30], for aesthetics, and general applicability to other tasks.

2.6.4 Genetic Algorithms

A genetic algorithm (GA) is a technique for search and optimization that is based on evolutionary principles. GAs represent a point within a search space using a chromosome that is based on a handcrafted genetic code.[31] Each chromosome consists of a string of genes that together encode its location in the search space. For example, the parameters of an AI agent can be the genes and a particular combination of parameters a chromosome. All combinations of parameters will represent the search space. By maintaining a population of chromosomes, which are continually mated and mutated, a GA can explore search spaces by testing different combinations of genes that seem to work well.[32] A GA is usually left to evolve until it discovers a chromosome that represents a point in the search space that is good enough. GAs outperform many other techniques in search spaces that contain many optima and are controlled by only a small number of parameters, which must be set by trial and errorcite [31].

Game Example: Genetic algorithms are very good at finding a solution in complex or poorly understood search spaces. For example, your game might have a series of settings for the AI, but because of interactions between the settings, it is unclear what the best combination would be. In this case, a GA can be used to explore the search space consisting of all combinations of settings to come up with a near-optimal combination[32]. This is typically done offline since the optimization process can be slow and because a near-optimal solution is not guaranteed, meaning that the results might not improve gameplay.

2.6.5 Filtered Randomness

Filtered randomness attempts to ensure that random decisions or events in a game appear random to the players. This can be achieved by filtering the results of a random number generator such that non-random-looking sequences are eliminated, yet statistical

randomness is maintained. For example, if a coin is flipped eight times in a row and turns up heads every time, a person might wonder if there was something wrong with the coin. The odds of such an event occurring are only 0.4 percent, but in a sequence of 100 flips, either eight heads or eight tails in a row will likely be observed. When designing a game for entertainment purposes, it is desirable for random elements to always appear random to the players. The technique involves keeping a short history of past results for each random decision that should be filtered. When a new decision is requested, a random result is generated and compared to the history. If an undesirable pattern or sequence is detected, the result is discarded and a new random result is generated. The process is repeated until a suitable result is accepted. Surprisingly, reasonable statistical randomness is maintained despite deliberate filtering[33].

Game Example: Simple randomness filtering is very common in games. For example, if a character plays a random idle animation, often the game will ensure that the same idle animation won't be played twice in a row. However, filtering can be devised to remove all peculiar sequences.

For example, if an enemy can randomly spawn from five different points, it would be extremely undesirable for the enemy to spawn from the same point five times in a row. It would also be undesirable for the enemy to randomly spawn in the counting sequence 12345 or favor one or two particular spawn points in the short term, like 12112121. Although these sequences can arise by chance, they are neither intended nor anticipated when the programmer wrote the code to choose a spawn point randomly. By detecting and filtering undesirable patterns or sequences with simple rules[33], a particular random decision can be guaranteed to always appear fair and balanced in the short term while still maintaining good statistical randomness.

2.6.6 Emergent Behavior

Emergent behavior is behavior that wasn't explicitly programmed, but instead emerges from the interaction of several simpler behaviors. Many life forms use rather basic behavior that, when viewed as a whole, can be perceived as being much more sophisticated. In games, emergent behavior generally manifests itself as low-level simple rules that interact to create interesting and complex behaviors.

Some examples of rules are seek food, seek similar creatures, avoid walls, and move toward the light. While any one rule isn't interesting by itself, unanticipated individual or group behavior can emerge from the interaction of these rules.

Game Example: Locking is a classical example of emergent behavior, which results in realistic movement of flocks of birds or schools offish[34].

However, emergent behavior in games is more commonly seen in city simulations, such as the ambient life in the Grand Theft Auto series. The city's inhabitants, composed of pedestrians, cars, taxis, ambulences, and police, create complex behavior from the interactions of agents using simple rules. Just like an ant colony exhibits large-scale

behavior from the actions of individual ants, a city is a complex system that emerges from the behavior of individual agents.[35]

2.6.7 Neural Networks

Neural networks are complex nonlinear functions that relate one or more input variables to an output variable. They are called neural networks because internally they consist of a series of identical nonlinear processing elements (analogous to neurons) connected in a network by weights (analogous to synapses). The form of the function that a particular neural network represents is controlled by values associated with the weights of the network. Neural networks can be trained to produce a particular function by showing them examples of inputs and the outputs they should produce in response. This training process consists of optimizing the network's weight values, and several standard training algorithms are available for this purpose. Training most types of neural networks is computationally intensive[36], however, making neural networks generally unsuitable for in-game learning. Despite this, neural networks are extremely powerful and have found some applications in the games industry.

Game Example: In games, neural networks have been used for steering racecars in Colin McRae Rally 2.0 and the Forza series, and control and learning in the Creatures series. Unfortunately, there are still relatively few applications of neural networks in games, as very few game developers are actively experimenting with them[37].

2.6.8 Reinforcement Learning

Reinforcement learning (RL) is a powerful machine learning technique that allows a computer to discover its own solutions to complex problems by trial and error. RL is particularly useful when the effects of the AI's actions in the game world are uncertain or delayed. For example, when controlling physical models like steering an airplane or racing a car, how should the controls be adjusted so that the airplane or car follows a particular path and What sequences of actions should a real-time strategy AI perform to maximize its chances of winning. By providing rewards and punishments at the appropriate times, an RL-based AI can learn to solve a variety of difficult and complex problems.[38]

2.7 Behavior trees vs. finite-state machines (FSM)

In theory, it is possible to express anything as a BT, FSM, one of the other abstractions, or as plain code. However, each model has its advantages and disadvantages in its intent to aid design on a larger scale. Specific to BTs vs. FSMs, there is a tradeoff between modularity and reactivity. Generally, BTs are easier to compose and modify while FSMs have their strength in designing reactive behaviors.

2.8 Conclusion

Game AI is distinctively different from many other related AI fields. The goal is to create intelligent opponents, allies, and neutral characters that result in an engaging and enjoyable experience for the player. Ultimately, the goal is not to beat the player, but rather to lose in a fun and challenging way.

Most games are populated by agents who sense, think, and act independently. however, even a single opponent can be thought of as an agent. Advanced agents might also learn and remember to present a deeper challenge. It is important to realize that whatever an agent senses, thinks, or remembers, is completely invisible and inconsequential to the player unless the agent can express the result through actions. An agent's outward appearance through movement, manipulation, animation, and dialogue is critical to making the agent appear intelligent. Typically, this requires tight integration and collaboration with the people who generate the art assets.

One of the most enduring techniques for endowing intelligence on agents is the ubiquitous finite-state machine. This simple computational model allows complex expertise to be expressed in a simple, easy-to-understand manner that is also convenient to debug. The actions and mindsets of an agent eloquently map to the states of an FSM, further allowing for simple, yet effective modeling of behavior. With the many enhancements developed for FSMs, it is easy to understand why they have become so universal within AI game development.

Finally, there are dozens of common and promising techniques for adding intelligence to games. Each game is unique and might require mixing and matching several different techniques. There is no single solution, and the resulting design is highly dependent on the exact requirements of the game. Therefore, a developer must become familiar with a broad range of techniques to experiment and make intelligent implementation decisions.

Chapter 3

System conception

3.1 Introduction

My goal is to create a game that uses artificial intelligence, coordinated in this chapter we will introduce the basic and advanced concepts of the game industry. Then using Concept map we will collect raw system information. At this point, We will also outline the system, as well as interact with the system and I'll finish with a conclusion.

3.2 The basic structure of the game engine

The basic concepts of the game industry start with the basic structure of the game engine program, in the sense of more precisely the game engine, which represents the nucleus of the system and controls all the component parts of the game, To better understand game basic structures, we have to think of a game system. Imagine there are various sub-parts that this system controls[50]. To picture this better, you can examine the diagram below :

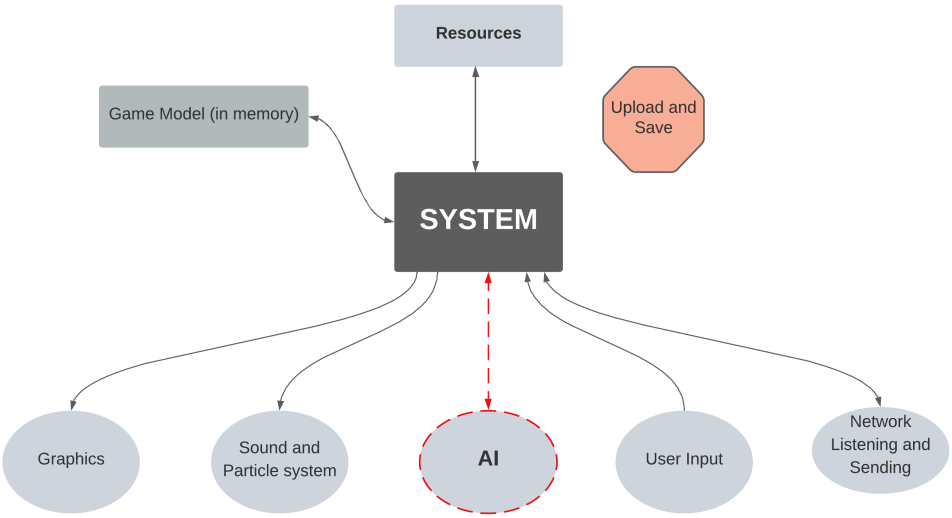


Figure 3.1: The basic structure of the game engine program

I will describe the various pieces that make up the game mechanics in the next few sections below[50].

3.2.1 System (Game system)

As you can see in the diagram above, the game system takes on the following tasks:
- Handles the loading and saving of files.
- It then enables the intro animation to run using the audio and graphic elements.
- While the game interface is being used, user input ensures that the graphics and sound parts work in harmony.
- Finally, the game system ensures that the graphics, sound, user input, AI and network parts work in harmony in the game.

3.2.2 Resources

The Resources describe the necessary files for the game located on the hard disk.
- These resources contain configuration files that describe the default settings in which the game will run.
- If available, it contains the video file for the intro that will be played when the game first starts.
- Contains files where the user gets information about the game before the game begins.
- It contains the graphics, music and chapter information in the game, alongside every other element the game will require to display audio and visual data to the player[50].

3.2.3 Game model

It includes the following:
- The game model is the state of the game graphics, music and text files loaded into memory from the resource.
- Image files are stored in memory or in the texture memory of the video card.
- Section, score, etc. information is kept in variables and data structures within the game model.

3.2.4 Upload and Save

This is the part that takes the graphics, music and other information from the source and puts it into the memory (game model) and undertakes the process of saving it back to the source in case of changes in scores, when the game is saved – or the save state – or when the game is recorded.
- Upload: Configuration files, chapter files, image files, music and sound files, 3D model files, video files, and help files.
- Save: Score files and other documents to be saved during the game.[50]

3.2.5 Graphics

This is the most important and hardest part of the game in our opinion. Because it is impossible to have a game without visuals (unless you are creating an old school text-based video game). This part is responsible for painting the screen and the following:
- Playing any video.
- Displaying the game interface.
- Viewing and displaying scores, help and configuration information.
Displaying the game's own graphics and visual elements.

3.2.6 User input

A game without interaction would be like watching a movie. User Interaction is one of the sines qua non of game mechanics that connects users to the game. The system is constantly interacting with the user. Keyboards, mice, cameras, and so forth are all capable of taking in user input. User input is the part of the game mechanic where we're going to use the hardware and equipment that lets the user interact with the game. User input allows a player to:
- Use the Game Interface.
- Review Help Files.
- Review Score Files.
- Configure the system and game-related features.
- Control the user's character in the game.

3.2.7 Sound and particle system

If you truly want to understand the importance of sound and music in a video game, try turning off the sound while playing your favorite game. You will notice that the value of the game drops significantly when there is no sound. To me, the value of the sound is as valuable as the graphics of the game.
- Music playing in the background.
- Sound effects for collisions and movements.

3.2.8 Artificial intelligence (AI)

Let's consider the intelligence level of the enemies in the game. The more humanoid they look and the more intelligent the behavior, the more fun the game will be. It doesn't just have to be the enemy either. In an open world game, citizens passing by look at their wristwatches, wipe the sweat on their foreheads in sunny weather, give way to you, even argue with you, attack you when you treat animals badly, and so forth. All of these elements can fall into the category of artificial intelligence.

3.2.9 Network listening and sending

Network listening and sending is the part of the game mechanic that allows the game to be played by more than one person, also known as multiplayer or online play. In this part,

the parties send information to the game server and the information received by listening to the server is updated in the game. Network listening and sending is responsible for:
- Establishing the Session: The players who will participate in the game agree on which protocol and on which port they will communicate at first.
- Packet Delivery: The moves made in the game are sent to the server in packets. Then the server distributes it to all parties after making the necessary change.
- Package Pickup: Simultaneously, the parties are aware of the changes made by listening to the server and reflect this to the game.
[50]

3.2.10 Hardware

Gamers love the various elements of a game and are always looking for better graphics, music, physics and artificial intelligence. For all of this to happen (and for games to get more complex over time), one of two conditions must be met:
Software optimization should be practiced and new algorithms should be developed. Appropriate equipment and hardware should be used, including increasing processing speed and memory.
Today, new algorithms and code optimization discoveries progress much slower than the speed at which new hardware develops. To combat this, instead of new algorithms, the developer should focus on issues such as reusability, code openness, and code security (encapsulation). To facilitate this, the hardware tries to increase the processor speed and memory capacity as much as possible for these new requests.
Many different categories of hardware can fall into this section, including the following:
- Graphics cards.
- Processors.
- Motherboards.
- Sound cards.
- Monitors, speakers and other input devices.

3.3 The general structure of the system

The general structure of the system is a functional view of the system architecture. By the concept map, We will be in constant contact with the system to define its limits, and thus avoid straying too far from the real needs of the final video game.

In this part, we will define the basic pillars of the game with a simplified model, and the part dedicated to artificial intelligence will be part of it, with the choice of the types of agents that we will design. After defining the types of agents, we will choose an approved model for building agents.

3.3.1 Concept Map

A concept map is a diagram that illustrates the relationship between concepts. It is a graphical tool that is often used by graphic designers, engineers and architects to

structure and organize knowledge. Concept maps typically depict ideas and information as circles or boxes, which are connected with labeled arrows in a hierarchical structure that is downward-branching. The relationship between concepts can be articulated using linking phrases such as “causes,” “requires,” or “contributes to.”

Concept map refers to a visual organizer that can enrich students’ understanding of a new concept. It is similar to brainstorming and mind mapping, since it challenges students to articulate the essential concepts or ideas. However, unlike brainstorming and mind mapping, concept mapping defines how these essential components relate to each other. It results in maps that are structured and complex, but also more informative.[51] A concept map is a diagram that shows the relationships between different ideas. This helps you understand how they’re connected.

Every concept map — whether it’s simple or complex — is made up of two key elements:

Concepts : These are typically represented by circles, ovals, or boxes and are called “nodes”.

Relationships : These are represented by arrows that connect the concepts, and the arrows often include a connecting word or verb (but they don’t have to). These arrows are called “cross-links”.[51]

3.3.2 General Architecture

The image below represents the basic model of the AI that represents the part framed in red in the previous image:

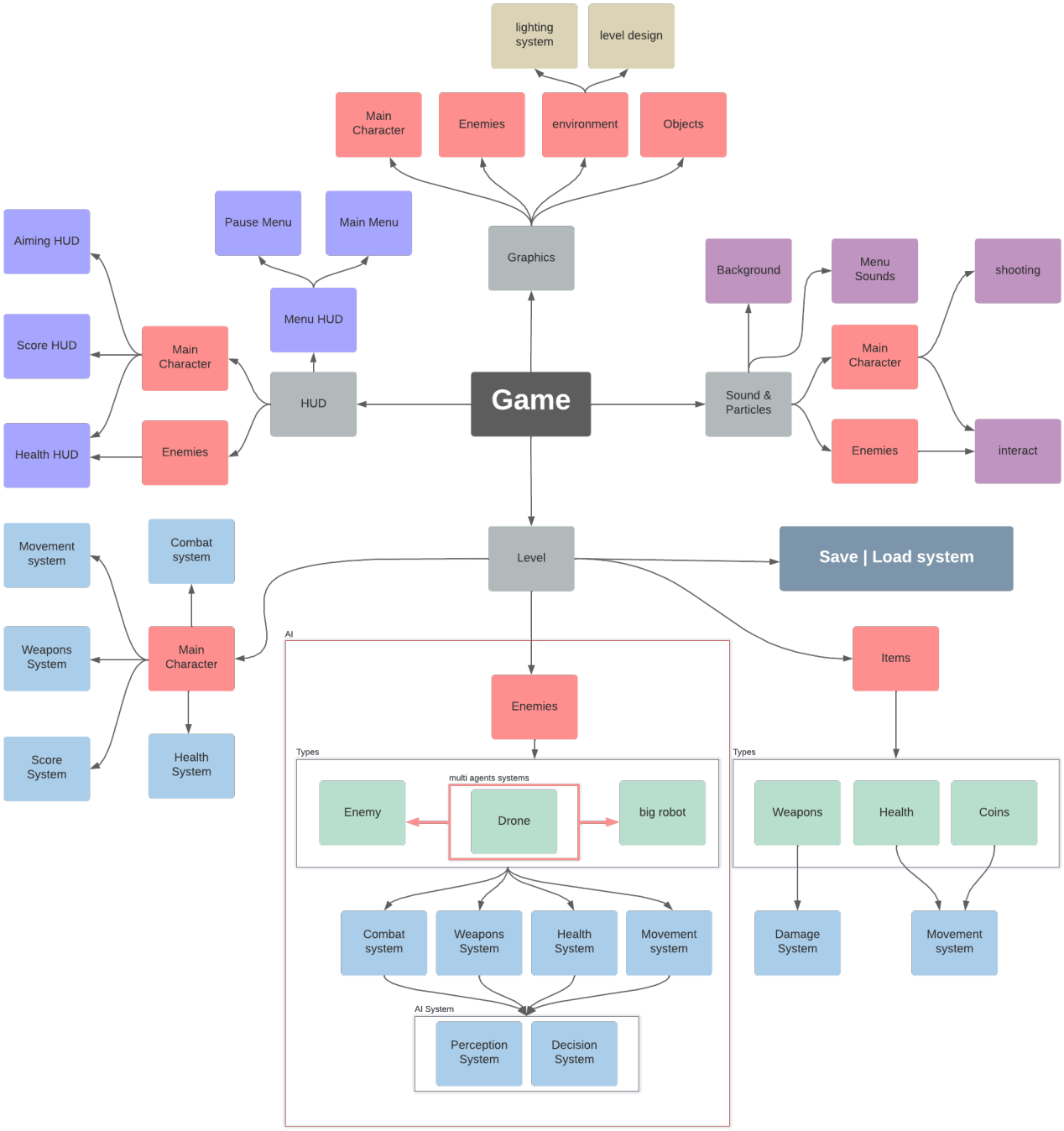


Figure 3.2: General Architecture of the game

I divided the general structure of the game system into 4 main branches:

3.3.2.1 HUD

The HUD (heads-up display) or status bar is the method by which information is visually relayed to the player as part of a game's user interface. It takes its name from the

head-up displays used in modern aircraft. The HUD is frequently used to simultaneously display several pieces of information including the main character's health, items, and an indication of game progression (such as score or level)[52].

3.3.2.2 Graphics

This section includes everything that is visible from materials, lighting, shadows, including designs (all models are included in this part) and the creation of environments in a general sense. This section contains the overall appearance of the game.

3.3.2.3 Sound and particle system

This section includes the sound system of the world and the characters, and the visual effects system responsible for adding vitality to the game.

3.3.2.4 Level

It is considered the most important and complex part that includes several systems that represent the game mechanics from the movement and combat system and AI systems that represent the enemies in general and the interactive systems of the game. AI is the important part of the level contains 3 types of agents::
- The first type, the less intelligent, does not have freedom of choice in movement.
- The second type (big robot) chooses random places for movement.
- The third type (the drone) uses a motion algorithm, which is an intelligent navigation system.

3.3.3 AI Architecture

The image below represents the basic components of the game [53]:

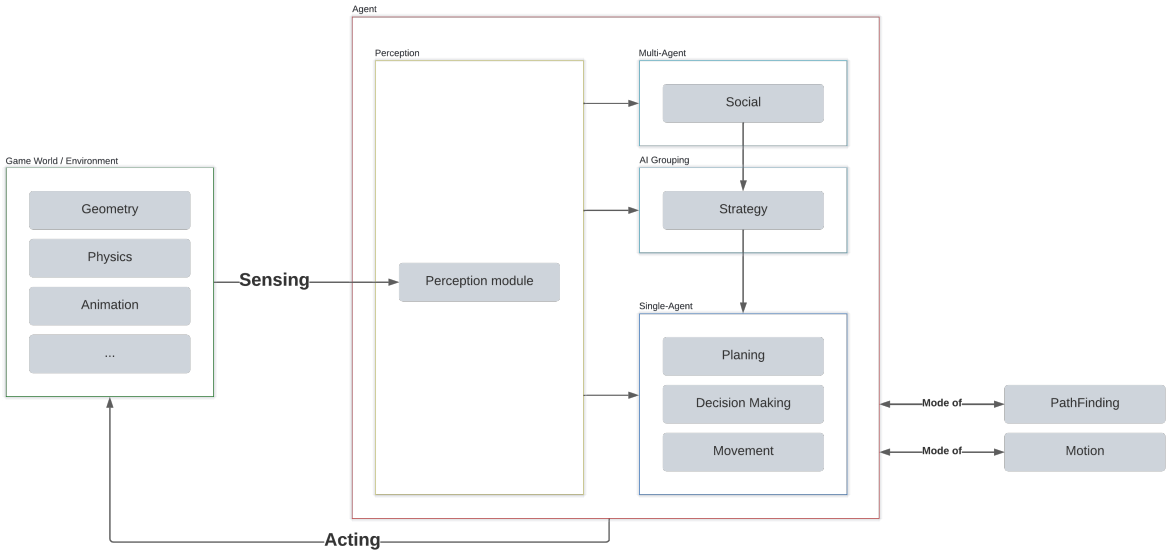


Figure 3.3: The AI architecture

This architecture system is divided into 3 main branches:

3.3.3.1 Game World / Environment :

This part represents the environment in which AI interacts. Which contains the main character who represents the target and the rest of the objects that are generally considered barriers, Add to that other physical factors such as gravity and others.

3.3.3.2 Agent :

It is divided into 4 main parts, each part has a special task, which is:

Perception : This part responsible for receiving information is by hearing, seeing, feeling, and others. The information is filtered and presented to the other parts to do their work.

Multi-Agent : This is the part responsible for communication between agents.

AI Grouping : It is the analytical part of the team where it develops strategies for agents to follow.

Single-Agent : This part represents the individual intelligence of each agent who is responsible for the individual decisions of the agent.

3.3.3.3 External Algorithms :

It is a set of algorithms that help an agent build and act upon decisions. Like algorithms to find the shortest path to the goal. Any algorithm can be used provided that it is effective for the system in particular.

3.4 Clarification of technical needs

After defining the general structure of the system, we need a model for determining the path and the steps that must be followed to create the different parts of the game system represented in the following model[54]:

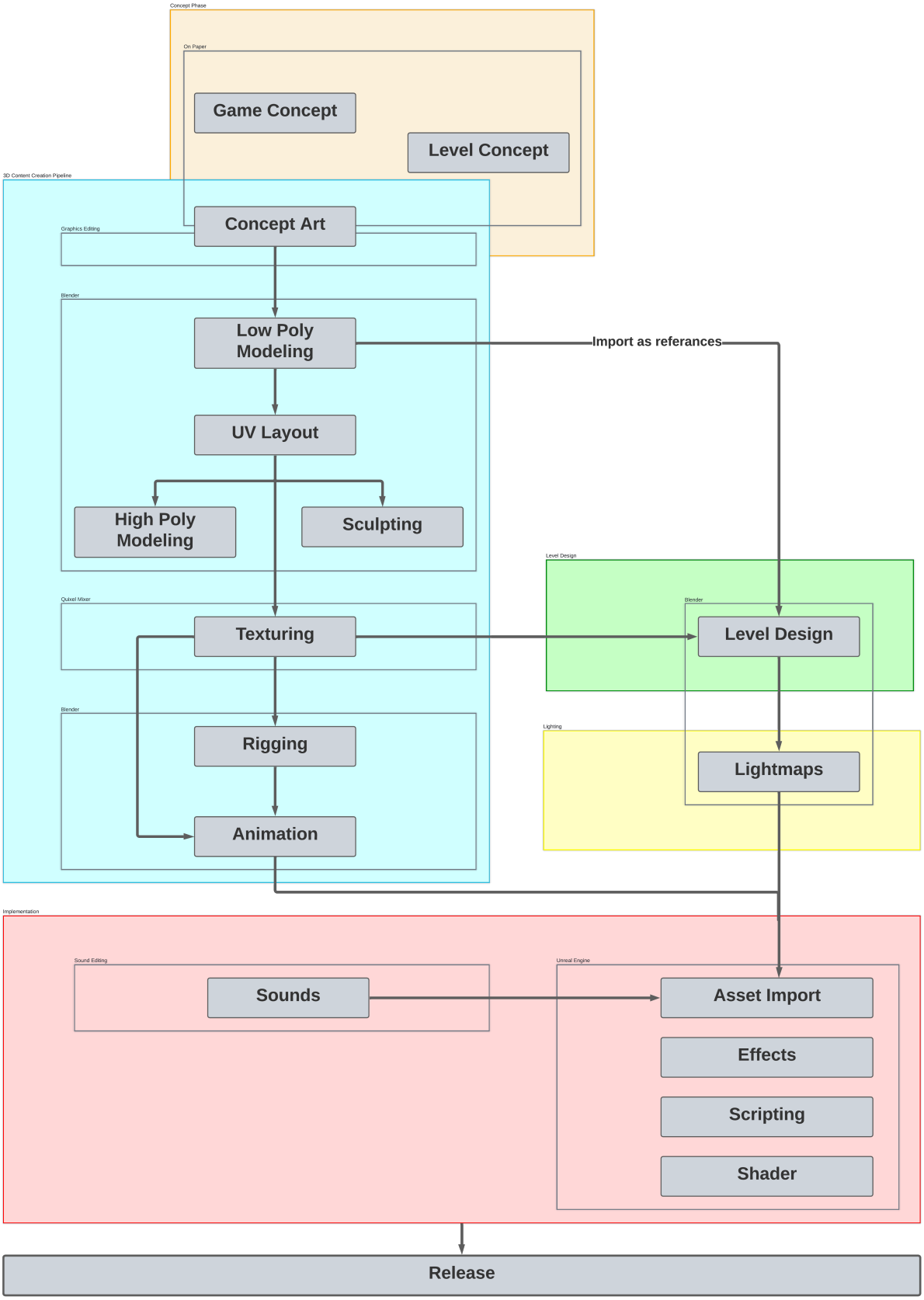


Figure 3.4: The game production pipeline architecture

3.5 The detail structure of the system

In this section, a more detailed scheme of the system is presented that shows the relationship between each part of the game :

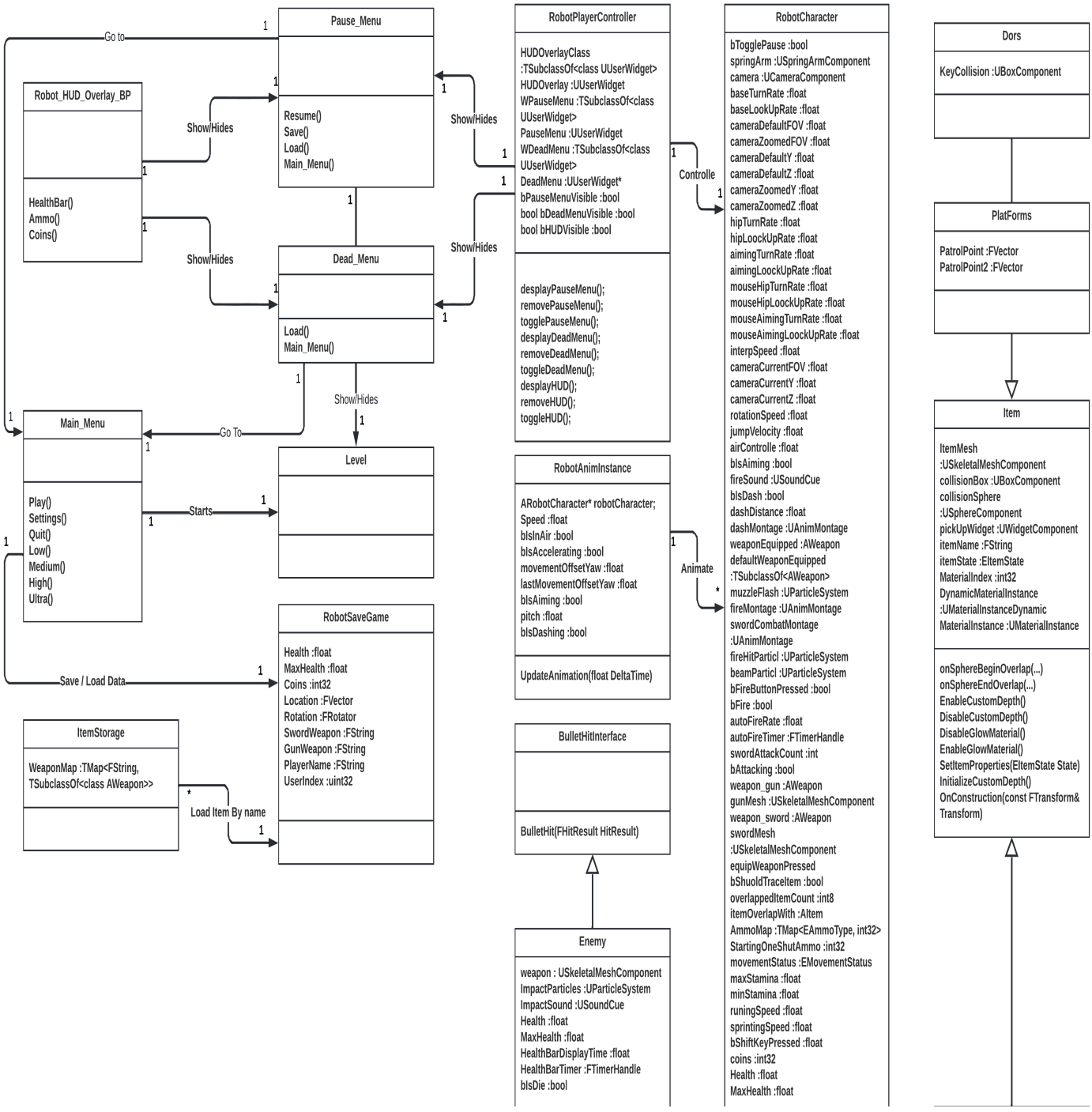


Figure 3.5: Game system simplified class diagram -upper half-

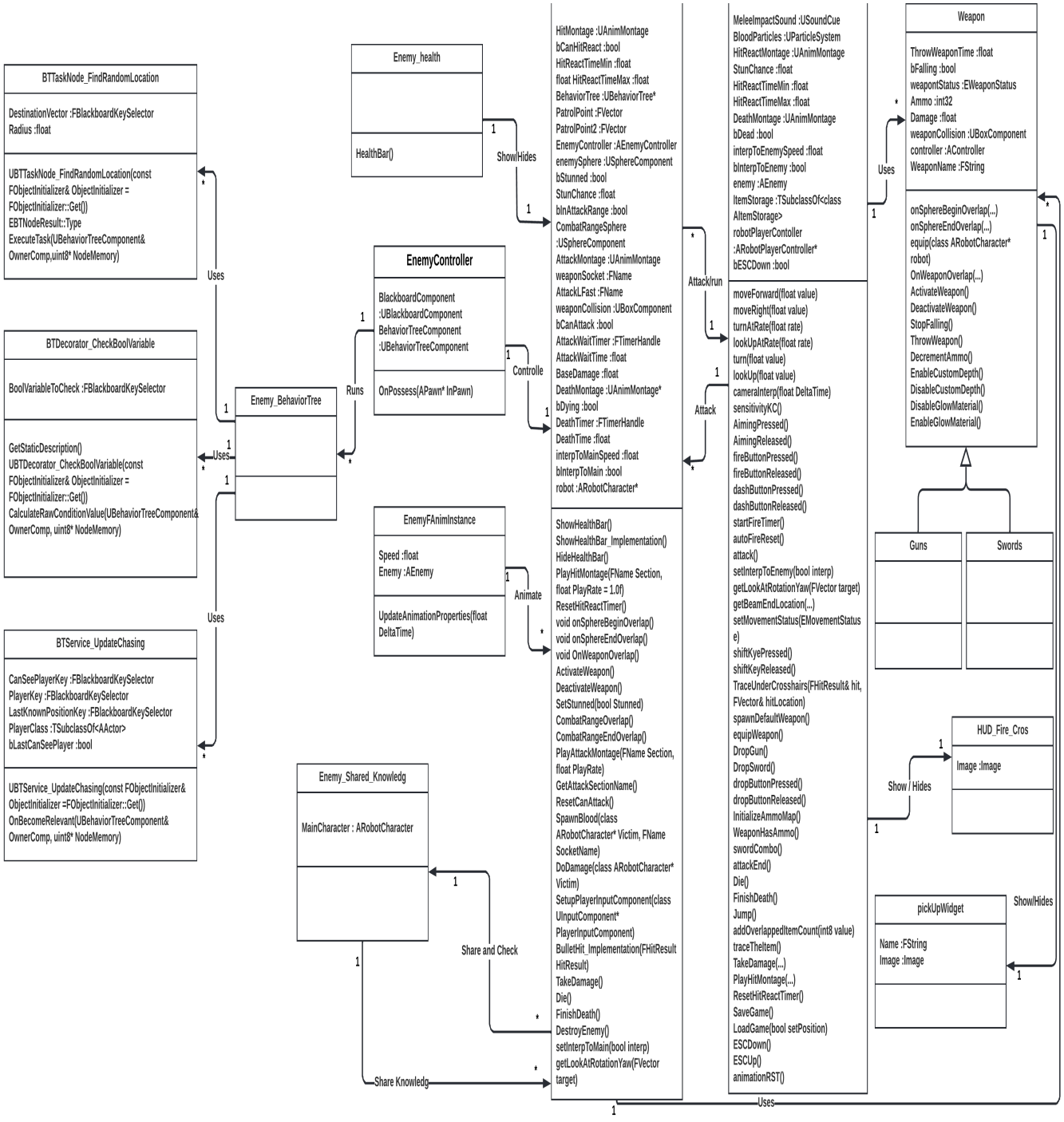


Figure 3.6: Game system simplified class diagram -lower half-

To make a diagram showing how AI works, we need to follow a specific scenario.

3.5.1 Scenarios of the game

My scenario is composed of 04 parts:

Search and roaming : The agent roams and searches for the main character by following pre-defined or randomly generated paths in the environment at this point the agent is using his senses to detect the main character.

stalking : When the main character is revealed, the client is chased after him using path-finding algorithms as well as avoiding barriers to achieve the desired goal.

Fighting : After reaching the target, the agent tries to eliminate the target using the available weapons.

Communication : It represents the communication system between multiple agents.

3.5.2 Conception of the artificial intelligence-based multi-agent system

the MAS is divided into two parts.

3.5.2.1 Conception of the agents' individual reasoning and decision-making

The planning and decision-making of individual enemies are modeled using behavior trees. All agents execute a copy of the same tree, but depending on their custom situation and perception of the environment they take different decisions and actions. Section 1.1 introduces behavior trees and Shows the main components of the behavior tree. The behavior tree of the game applies the three main situations or states described From before: Search and roaming attempting to detect the player, stalking trying to reach the player, Fighting to defeat the player, and doing tasks that are not directly connected with the player at that moment (since there is no information about the player's actions). In the game, there are three types of enemies, each enemy has its intelligence represented in the behavior tree, despite the great difference in the way each behavior tree works, the method of making it remains subject to the same laws.

Each behavior tree is broken down into different subsets, depending on whether the enemy can see the player character, whether they can't but have some other information that can be used to try to find the player character (any sound or visual stimulus that may be related to the player and that has an estimated or exact source location), or whether no relevant information about the player is available. This is checked by different decorator nodes. (Decorator nodes are all shown in blue, but they do not all act the same way. The mentioned ones are represented by a rhomboid shape, and they do not execute in a preestablished moment.

Instead, they periodically check a conditional expression to ensure that it is true; if not, they immediately abort the execution of their subtree and return false to the decorator's parent node. these nodes allow resetting the execution flow when something requires changing from one of the logical states (the one in which the player character is visible, the one in which they are not seen but there is some information that may help to find them or the one in which no relevant information is available) to another

one. This can happen after one of the updates of the service that checks perceptions, interruption related to hearing or communication is received. (service nodes are all shown in green color), For example, if an agent has no information about the player, the tree will be executing the idle subtree (e.g. patrolling in the random zone or patrolling regularly between two points), but if the player character is suddenly seen by the agent (i.e.detected by the periodically executing service node or enter the field of vision) the decorator that requires to be idle with no information about the player stops being true, so the flow of execution returns to the parent selector, which leads to entering the subtree that has the condition of being able to see the player character, that leads to trying to chase him, and upon reaching it the combat behavior tree activates to try to defeat the player. (All tasks performed by the behavior tree represent by the tasks nodes and are all shown in pink color).

Representation of the game behavior tree :

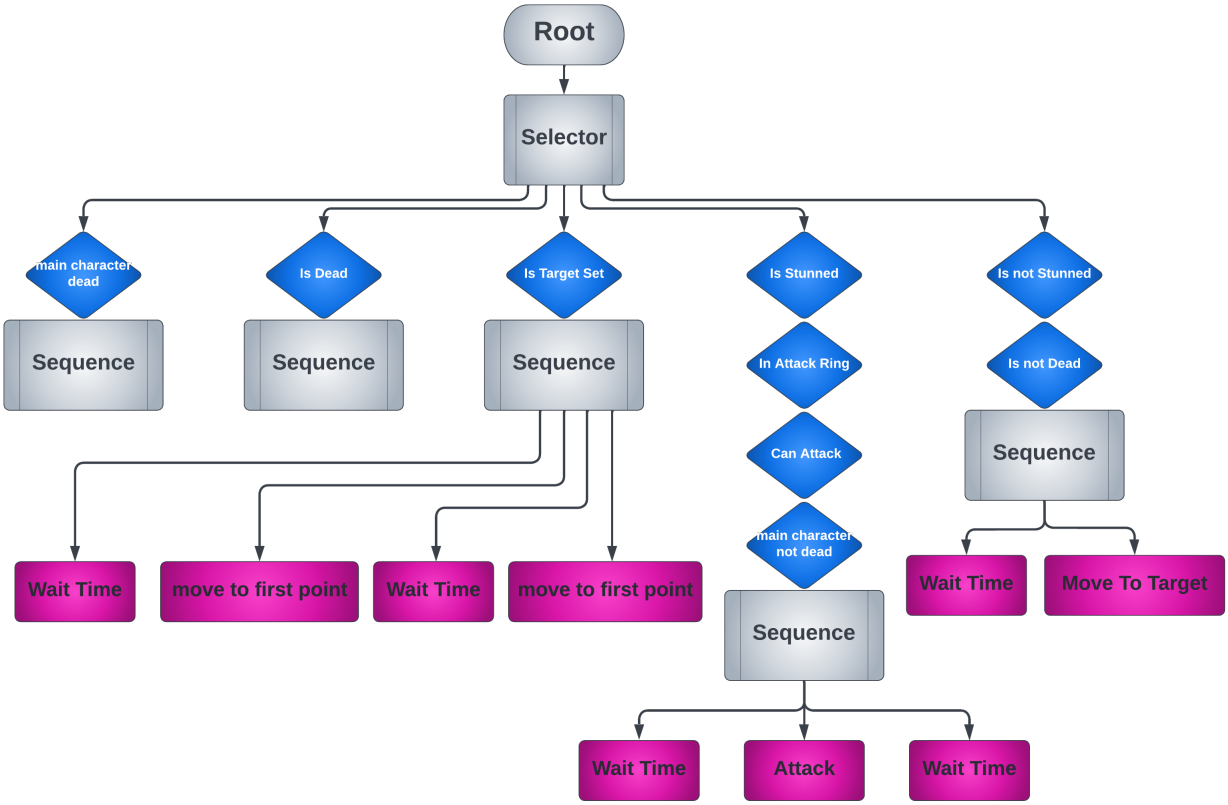


Figure 3.7: Representation of the first type of behavior tree for enemy characters

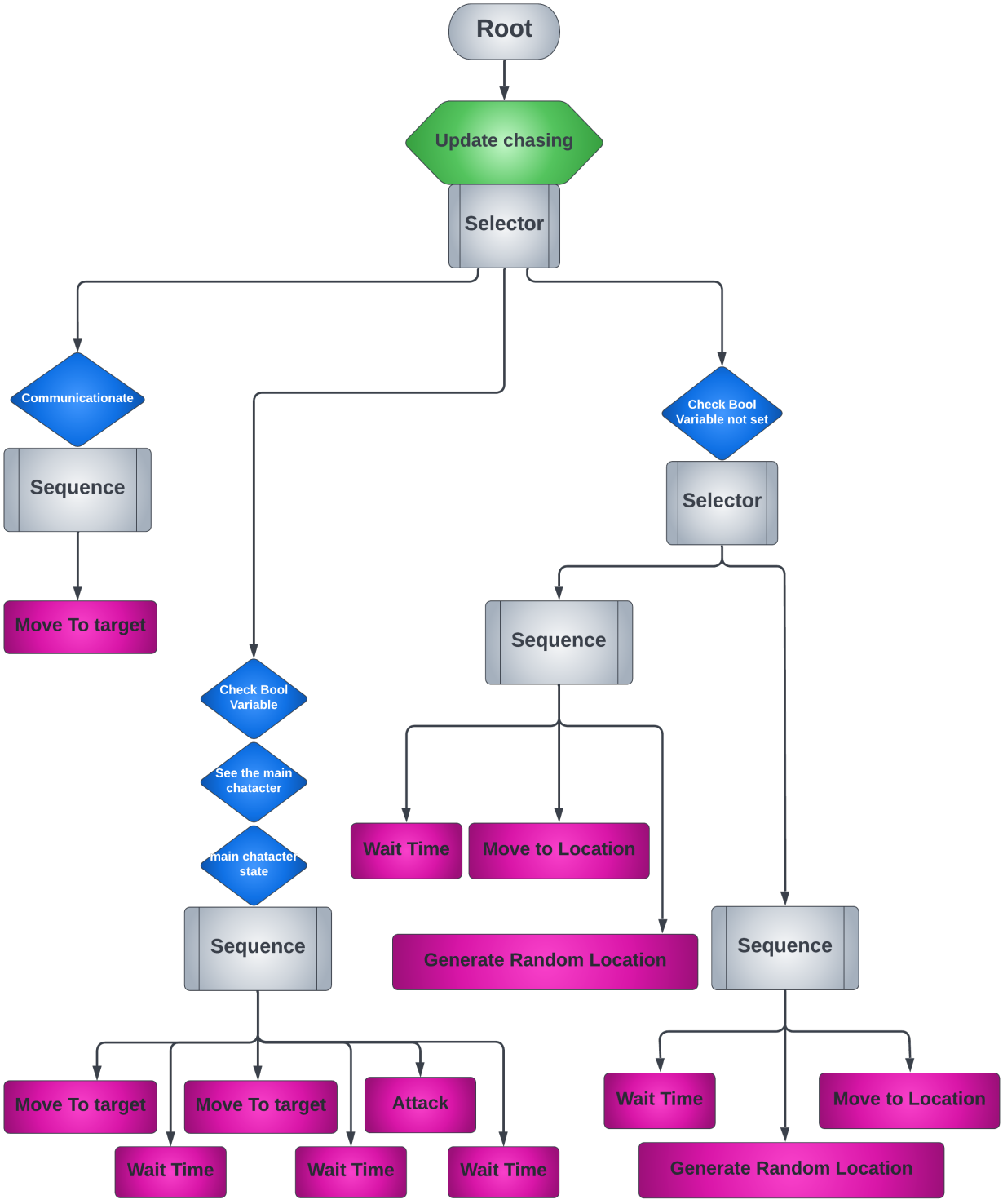


Figure 3.8: Representation of the second type of behavior tree for enemy characters

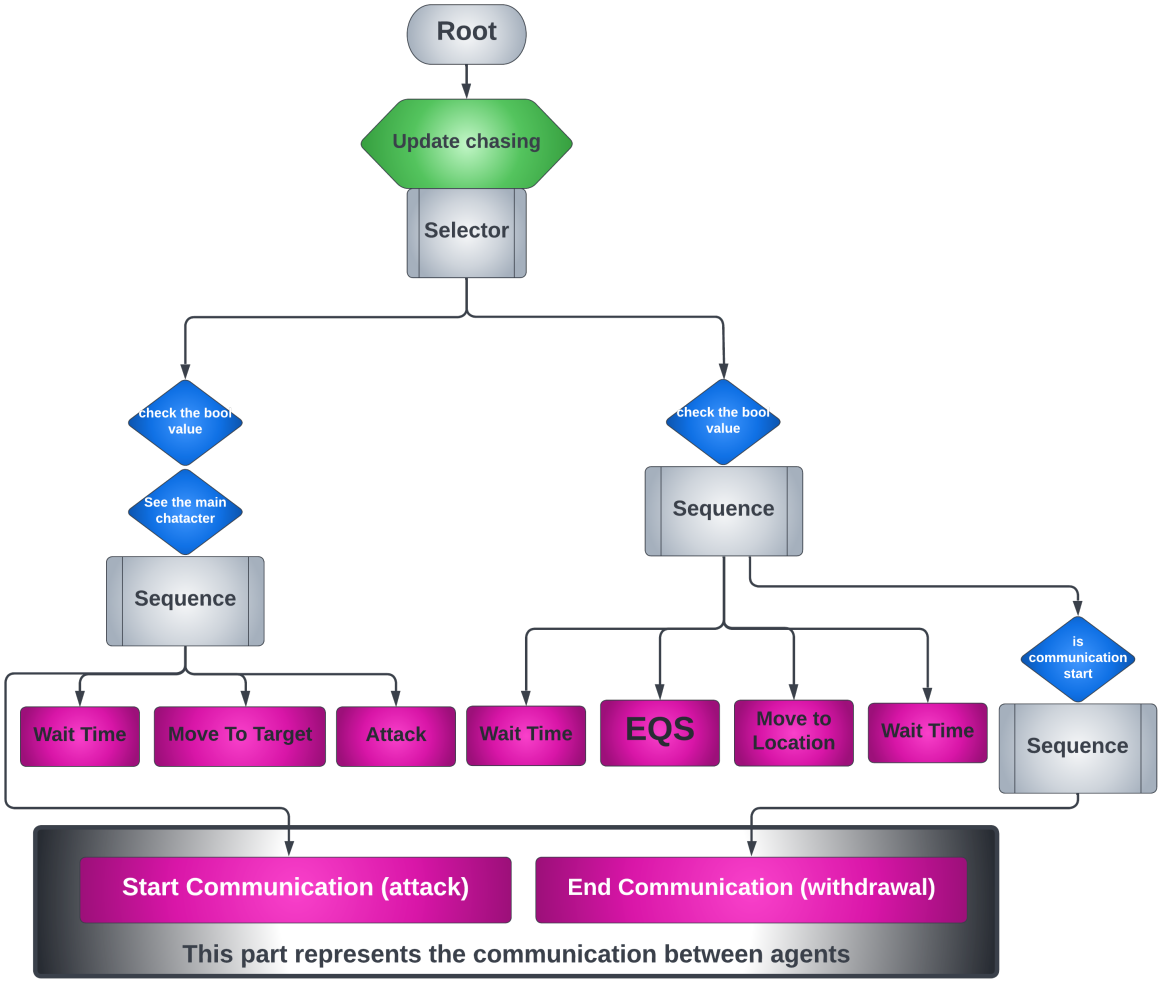


Figure 3.9: Representation of the third type of behavior tree for enemy characters

3.5.2.2 Conception of Multi-agent system analysis: communication between agents

All the enemy characters in the game form a single MAS. The objective of both individual agents and the system is to defeat the player character as fast as possible. One type of enemy can send information to all other agents (They have to be under his command) at any time: it should be thought of as radio communication between machines. Enemies examine the information they have received from leaders and decide how to use it: agents are independent, but they take into consideration What the leaders say.

In our game, two types of information can be communicated, and they are all related to the player's character. as shown in Figure 3.10, the first one is the Order to attack the main character when has just been seen, the orders updated with time, and the approximate location from where the sender agent thinks that the character's footsteps have been heard or was it last seen, the second one is the Order to withdraw When losing track of the main character Where all his followers must withdraw.

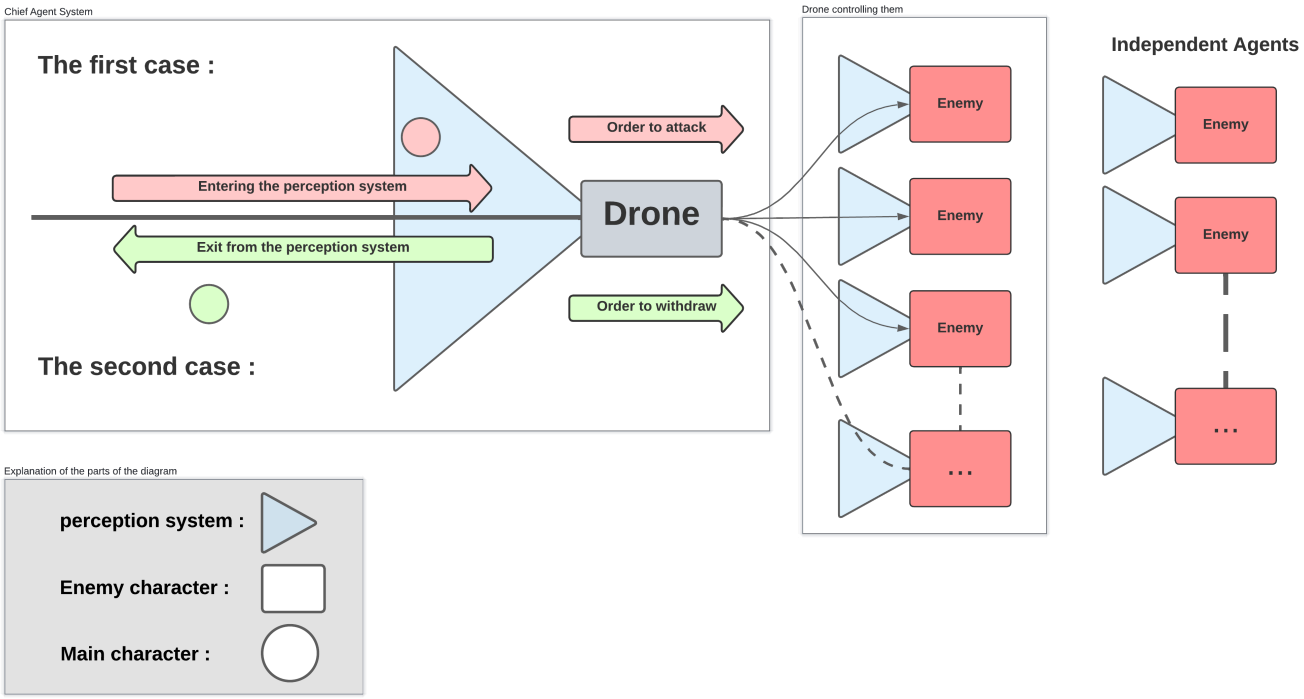


Figure 3.10: Representation of the Multi-agent system

3.6 Conclusion

In this chapter, We have given an overview of our game system, a game that uses a MAS to control agents. We started by showing the infrastructure of the game engine which is the key to understanding what We need, and then We detailed the general structure of the system: the base of the game, its overall architecture, then the infrastructure of the agent and MAS for the game AI system. Then put the stages We need to pass to end up with an integrated game based on artificial intelligence.

Chapter 4

System implementation

4.1 Introduction

I have almost come to the end of the development process, and in this chapter, We will make choices about programming. We start with the presentation of programming languages, and the development tools used in our Game. Finally, We end with the presentation of the agent system of our final Game.

4.2 Development languages used

In order to develop a game, one must master a combination programming languages.

I will describe the various pieces that make up the game mechanics in the next few sections below.

4.2.1 C++

Released in 1985, C++ is one of the best programming languages available. C++ is a highly optimized code that allows for greater memory management, a necessary attribute of high-poly, and high-definition video games. C++ programming allows for stylized gameplay and is a popular language for triple-A (AAA) titles, as well as indie games. C++ language runs with most game engines, making it one of the most common selections for game programmers.[55] Since C++ is a high-level language that will teach you the basics of object-oriented programming (OOP), it's a good idea to learn it. It is also the language used to create console and Windows games. Also, it uses OpenGL or a similar framework. C++ is a fast-compiling programming language. You also get a lot of say in memory management. It has extensive libraries useful for designing and powering complex graphics. There's a lot of literature for you to study and learn from, because it's been the video game programmer's language of choice for decades, and you'll find online communities ready and willing to answer your questions.[56]

It would be a lie if we said that C++ is easy to learn. It is difficult to learn compared to other programming languages. However, it can be useful not only because C++ games are easy to distribute on various platforms, but also because if you know C++ you can

quickly learn C# and other object-oriented languages, including C and Java (even though Java is not technically an OOP language). C++, C, Java, and C# are some of the most actively used programming languages today.[56]

To summarize, learning C++ is a good choice if you want to create video games from scratch for multiple platforms.

Figure 4.1: C++ logo

.

4.2.2 C#

Pronounced "C sharp" ,this popular programming language was released by Microsoft in 2000. C# is a relatively easy programming language to learn and is often used by smaller game studios. C# is another one of the main codes seen in popular game engines.[55] The benefit of C# for video game development lies in the XNA framework. This is a set of tools and workspaces by Microsoft that are particularly suitable for developing games on Xbox or Windows platforms.[56]

If you compare C++ and C#, you might consider this example: C++ is like a manual transmission car; C#, on the other hand, is like an automatic transmission car. Let's consider the Unity game engine. If you use the Unity game engine, you have to code your game scripts in C#. However, the core of this game engine was developed using C++ code.[56]

The platform you target, the game you want to make, the game engine you will use, etc. will affect the language you choose. No matter, however, learning C# for game development would be a great idea.[56]

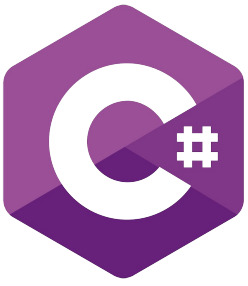


Figure 4.2: C# logo

.

4.3 Development tools

The general structure of the system is a functional view of the system architecture. By the concept map, We will be in constant contact with the system to define its limits, and thus avoid straying too far from the real needs of the final video game.

4.3.1 Sourcetree

Sourcetree is a free graphical user interface (GUI) desktop client that simplifies how you interact with Git repositories so that you can fully concentrate on coding. Say goodbye to the command line — this GUI makes it easy to visualize and manage your repositories. It also integrates with Mercurial to ensure an efficient, consistent development process. Visualize your work and execute push commands with a whole new level of confidence. Even changing or discarding a file, a hunk, or an entire line is now simple.[57]

Figure 4.3: Sourcetree logo

.

4.3.2 Blender

Blender is a free and open-source 3D computer graphics software tool set used for creating animated films, visual effects, art, 3D-printed models, motion graphics, interactive 3D applications, virtual reality, and, formerly, video games. Blender's features include 3D modeling, UV mapping, texturing, digital drawing, raster graphics editing, rigging and skinning, fluid and smoke simulation, particle simulation, soft body simulation, sculpting, animation, match moving, rendering, motion graphics, video editing, and compositing.[58]

Figure 4.4: Blender logo

.

4.3.3 Quixel Mixer

Quixel Mixer is primarily a texturing software that allows you to create and Utilize Physically Based Rendering (PBR) materials. Its biggest advantage is it allows you to procedurally create tileable textures and directly paint on 3d models utilizing a large free library of materials and the Quixel Megascans library.[59]

Figure 4.5: Quixel Mixer logo

.

4.3.4 Visual Studio

Visual Studio, also known as Microsoft Visual Studio and VS, is an integrated development environment (IDE) for Microsoft Windows. It is a tool for writing computer programs, websites, web apps, and web services. It includes a code editor, debugger, GUI design tool, and database schema designer, and supports most major revision control systems. It is available in both a free "Community" edition and a paid commercial version.[60]

Figure 4.6: Visual Studio logo

.

4.3.5 Unreal Engine

Unreal Engine (UE) is a game engine developed by Epic Games, first showcased in the 1998 first-person shooter game Unreal. Initially developed for PC first-person shooters, it has since been used in a variety of genres of three-dimensional (3D) games and has seen adoption by other industries, most notably the film and television industry. Written in C++, the Unreal Engine features a high degree of portability, supporting a wide range of desktop, mobile, console, and virtual reality platforms.[61]

Figure 4.7: Unreal Engine logo

.

4.4 Implementation of the characters' abilities and interactions with the world

To be able to move in the world and interact with other game elements, characters use a set of components, which are the parts that make characters a functional whole. A capsule component is the invisible root element of the character, and it defines the base collision with the world. Its diameter and height values match the desired character dimensions. A mesh component attached to the root capsule gives characters a physical appearance and the ability to play different animations. A special type of Blueprint, an Animation Blueprint, is responsible for toggling animations when the characters' circumstances change (e.g. switching from an idle animation to a jumping one when the character starts jumping).Figure 4.8 shows the state machine of the Animation Blueprint used by the main character, Many conditions are tested to determine the exact animation that is played on different bones of the character's mesh in each state of the state machine.

For example, in the idle or running state, the legs are influenced by the current speed of the character: an idle animation should be played when speed is zero, while a running animation is a right choice when speed is high; the two animations are blended with different weights depending on speed. If the character has a weapon, it is checked whether the character is doing a specific action with it, to determine which animation should be played.

Figure 4.8: Detail movement system for the main character

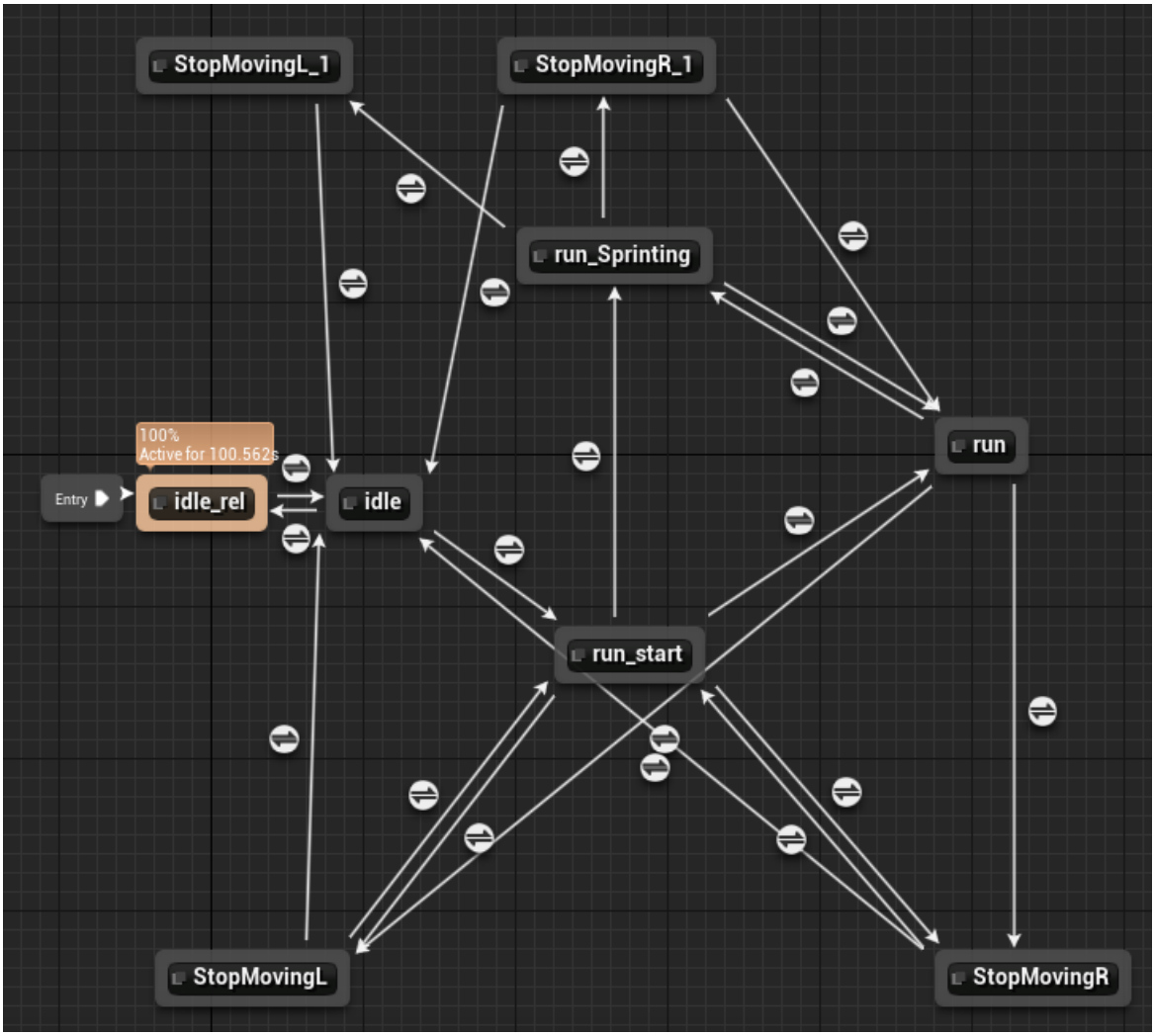


Figure 4.9: A part of the movement system of the main character

Characters also have a movement component that allows them to move. Unreal Engine has a type of movement component thought for anthropomorphic characters that allow to easily configure how characters should move. Among the parameters that can be set, one can find the maximum speed or the rotation rate of the character, for example. Other values include the maximum height that single steps can have to allow the character to go upstairs without the need of jumping. The ability to jump is precisely another of the built-in features of the characters' movement component, and parameters such as the jump speed can be set. Crouching is supported as well. Despite of the built-in characters' capabilities, it is the programmer's task to handle the animation transitions to be coherent with the change in actions, checking conditions in the Animation Blueprint (e.g. the transition to the jump animation in the state machine should only take place if the character is physically jumping).

The player character's physical movement is determined by the player's input, e.g. the character physically skates when the skate key or button is pressed (if the character is not already dashed or in a situation where skating is not possible).

4.5 Implementation of the AI based multi-agent system

4.5.1 the agents' individual reasoning and decision-making

In this part We will create the behavior trees that were designed in the previous chapter:

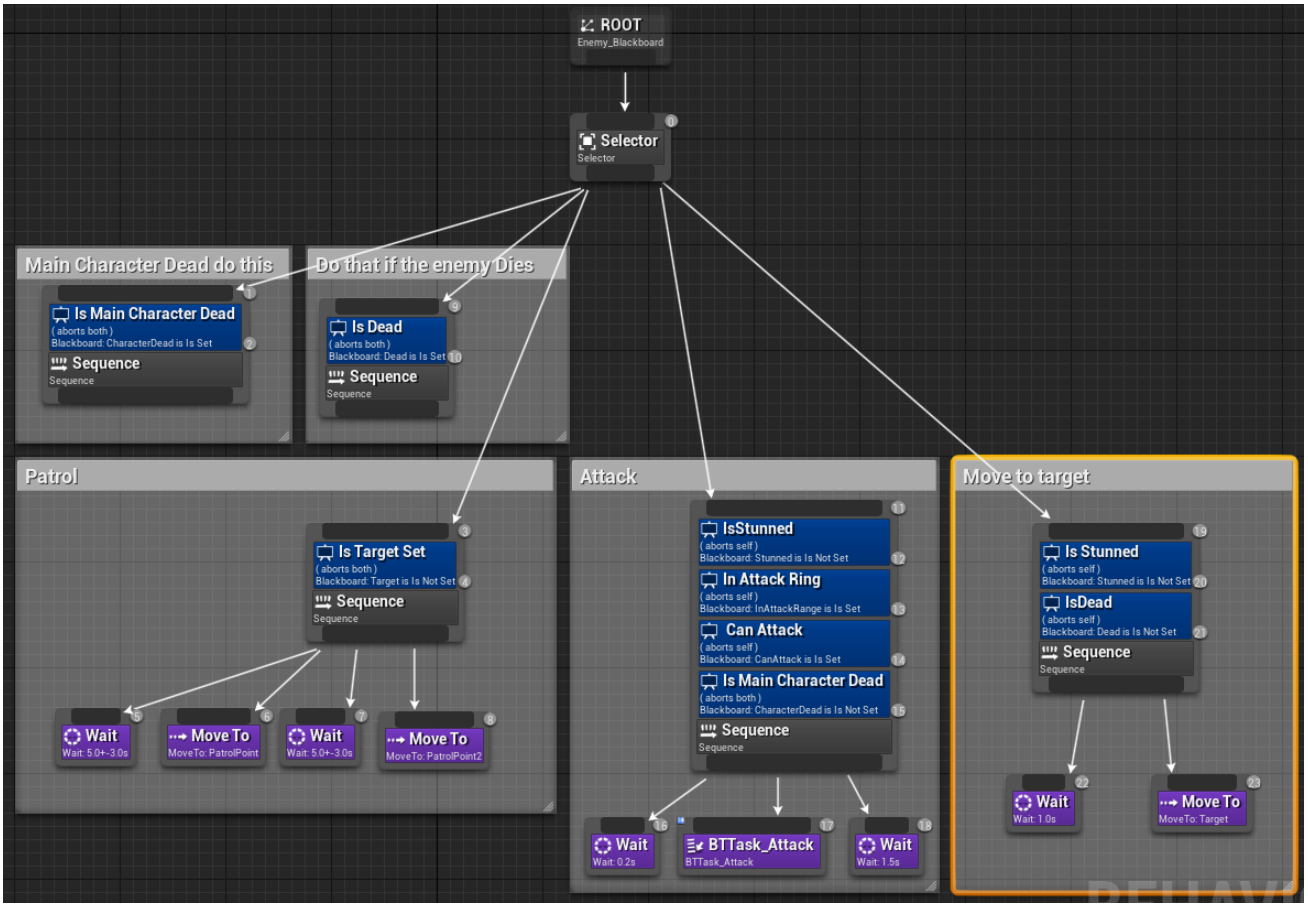


Figure 4.10: Implementation of the first type of behaviour tree for enemy characters, as seen inside Unreal Engine's behaviour tree editor

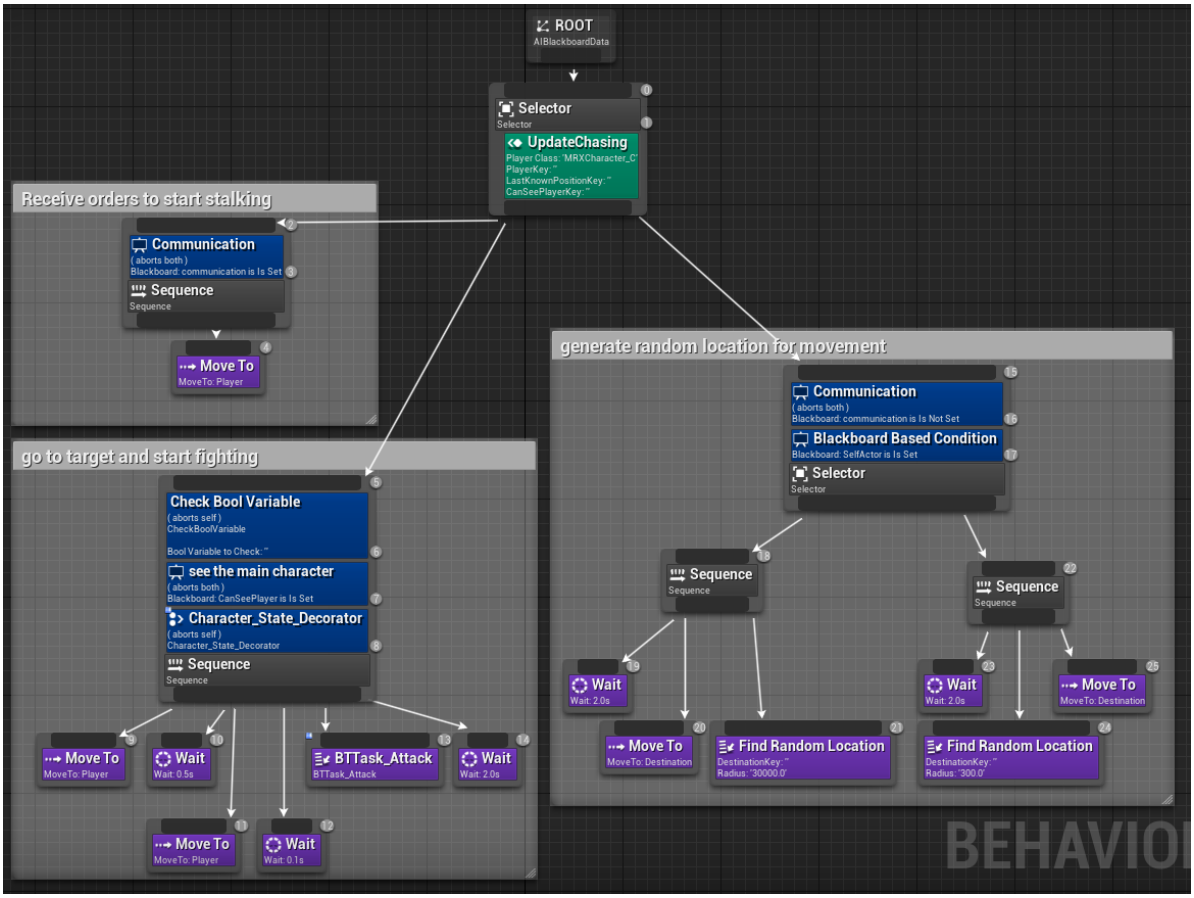


Figure 4.11: Implementation of the second type of behaviour tree for enemy characters

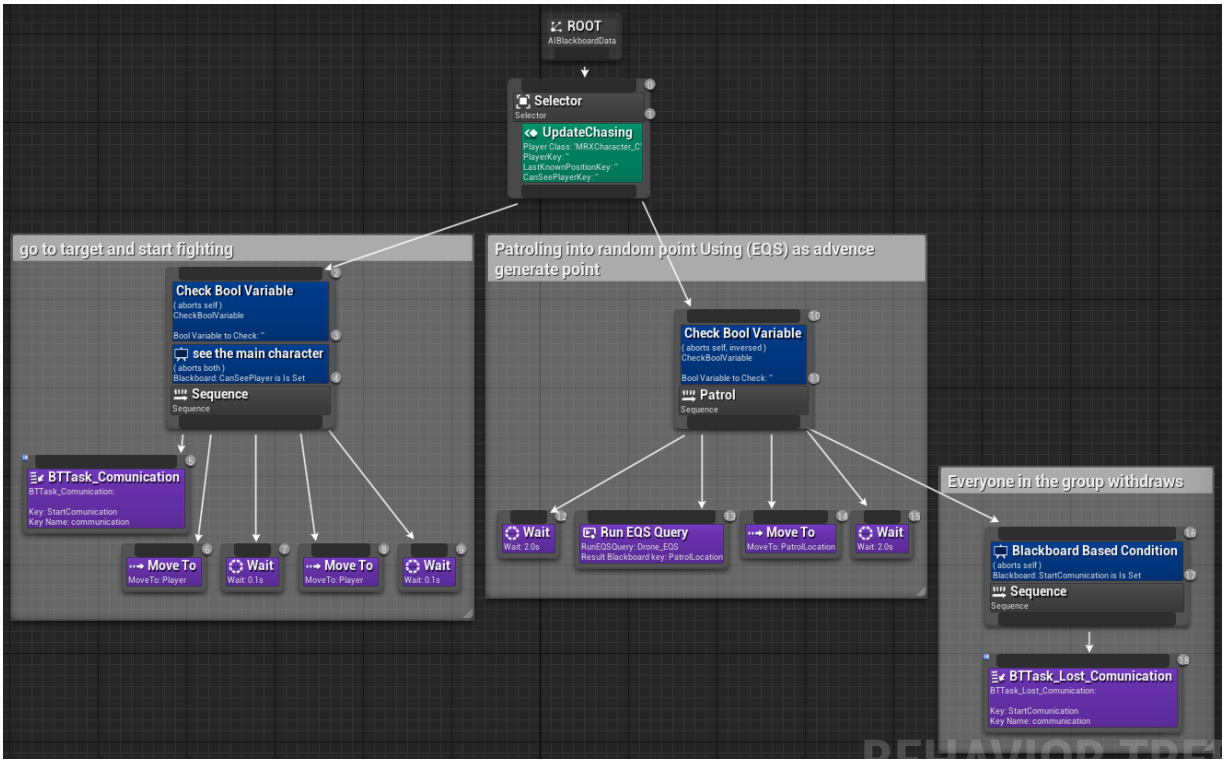


Figure 4.12: Implementation of the third type of behaviour tree for enemy characters (Drone)

This is an overview of enemies with the behavior tree system.

Figure 4.13: The first type of enemy characters

Figure 4.14: The second type of enemy characters

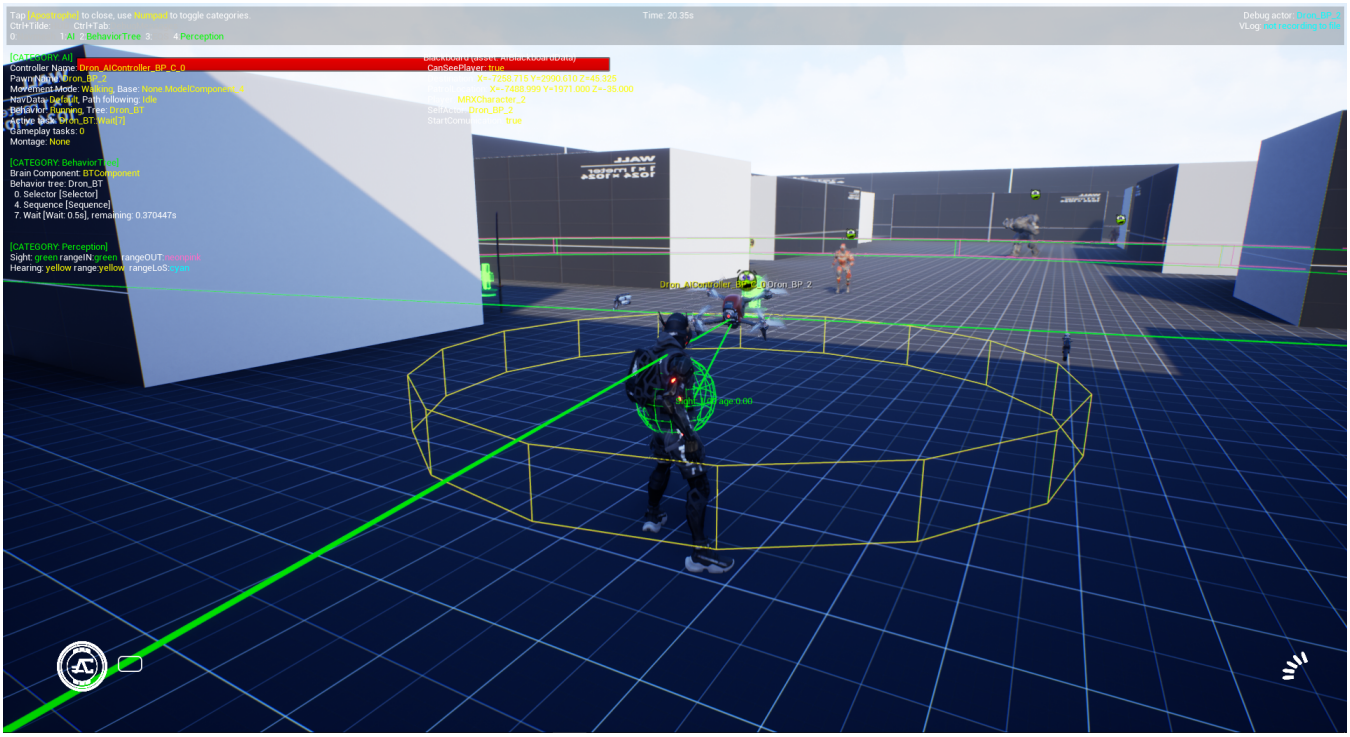


Figure 4.15: The third type of enemy characters (Drone)

4.5.1.1 EQS

The Environment Query System (EQS) is a feature within the AI system in Unreal Engine 4 (UE4) that is used to collect data from the environment. Within EQS, you can ask questions about the data collected through a variety of different Tests which produces an Item that best fits the type of question asked.

An EQS Query can be called from a Behavior Tree and used to make decisions on how to proceed based on the results of your Tests. EQS Queries are primarily made up of Generators (which are used to produce the locations or Actors that will be tested and weighted) and Contexts (which are used as a frame of reference for any Tests or Generators). EQS Queries can be used to instruct AI characters to find the best possible location that will provide a line of sight to a player in order to attack, the nearest health or ammo pickup, or where the closest cover point (among other possibilities). [62]

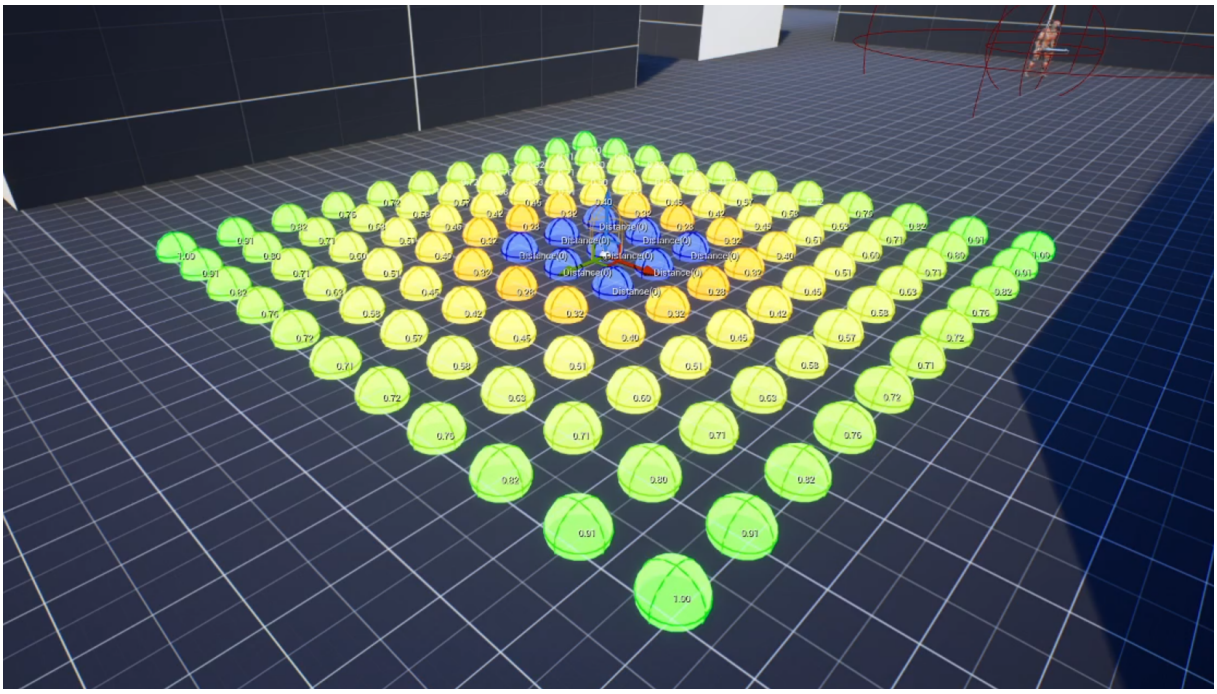


Figure 4.16: simple form for an Environment Query System

EQS algorithm of this game The use of the environmental query system gives the way enemies move more dynamic by discovering and bypassing barriers and moving in precise places.

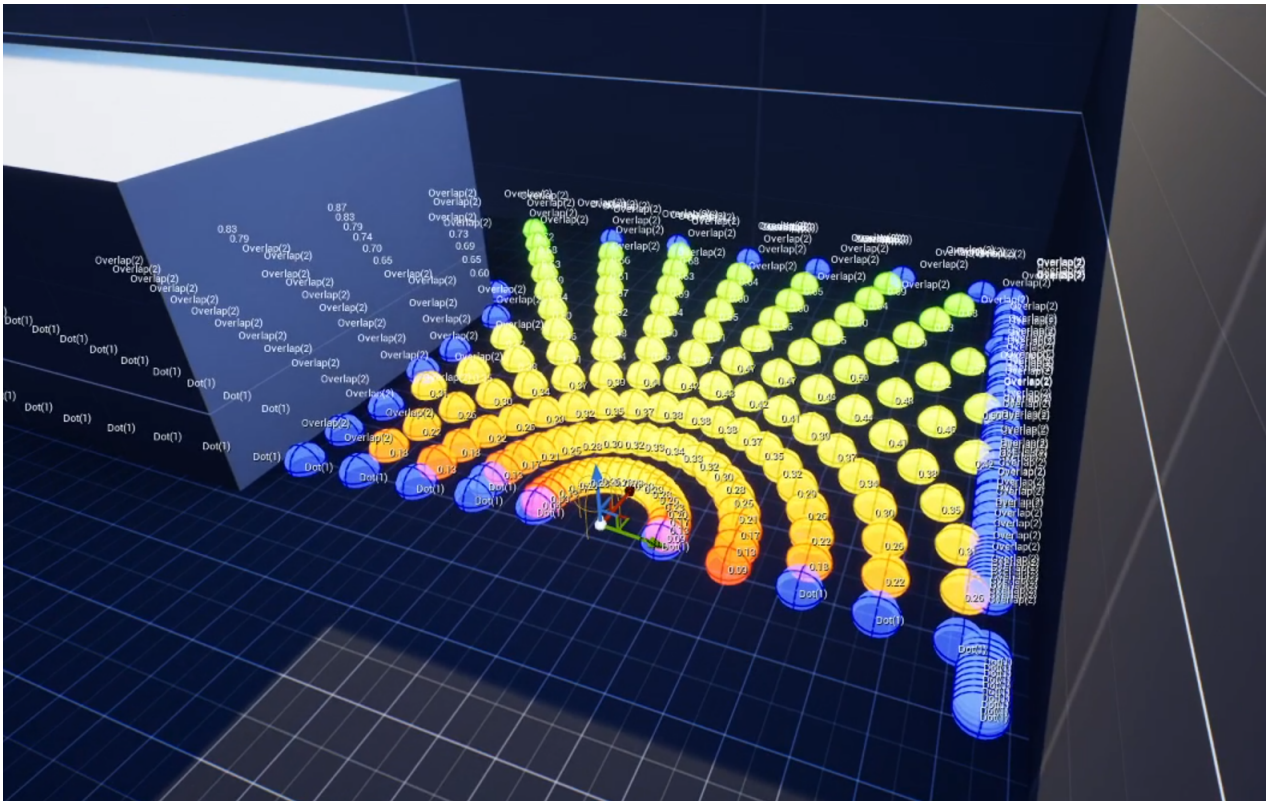


Figure 4.17: complex environmental query system used by the enemy (drone)

Where barriers such as walls and characters are defined and these places are ignored, places behind models are ignored and considered outside the calculation of the EQS algorithm.

4.5.2 Multi-agent system analysis: communication between agents

the communication system is a class of BT_Task node (execution nod), so in this part, orders are issued from the commander to the subordinates to begin the attack on the main character, this class starts executing when the perception system of the enemy detects the main character.

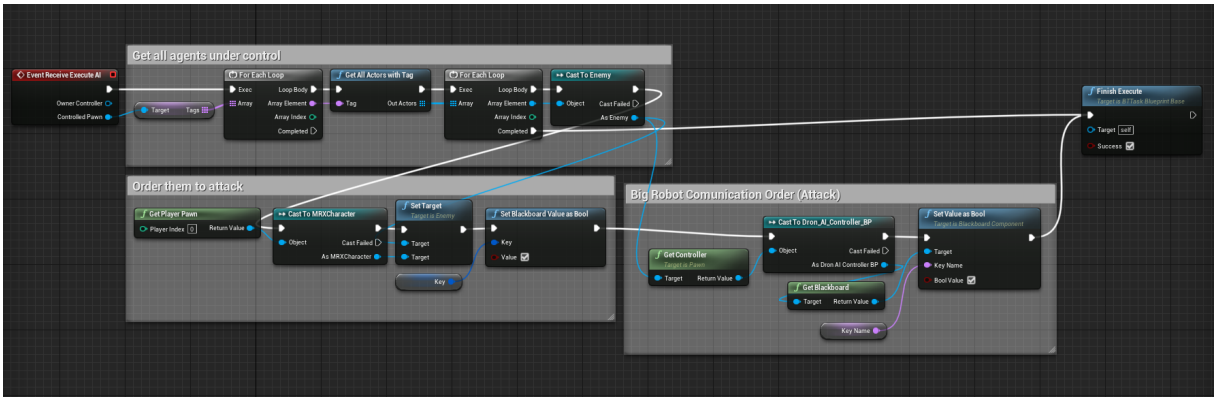


Figure 4.18: Implementation of the communication system of giving orders to agents to attack

On the other hand, orders are issued from the commander to the subordinates to withdraw from the battlefield and everyone returns to their position, and this class begins to be implemented when the enemy's perception system loses the main character (the main character moves out the perception system).

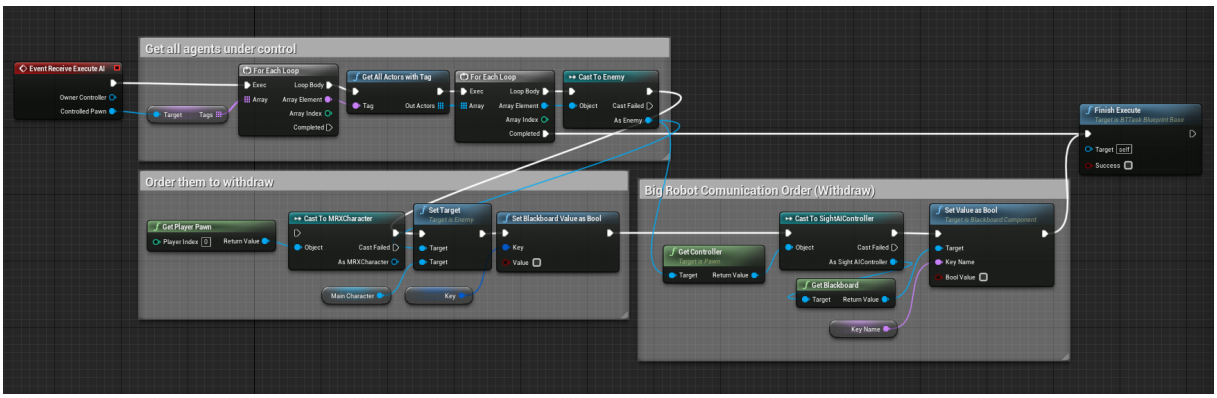


Figure 4.19: Implementation of the communication system to give orders to agents to withdraw to their centers (Drone)

4.6 Implementation of the in-game user interface

The user interface (UI) that is seen while playing the game is also called Heads Up Display (HUD). The elements that form it were designed using Unreal Motion Graphics (UMG), the engine's tool for creating widgets and other UI elements.

4.6.1 Main menu

The main menu HUD contains a simple widget containing buttons, play to start the game, settings for cheng the quality of the game, and quit for ending the game.

Figure 4.20: Implementation of the Main Menu

4.6.2 Pause menu

The pause menu HUD contains a simple widget containing buttons, resume resuming the game, save for saving, load For Load the last saved settings of the game, and main menu for return to the main menu.

Figure 4.21: Implementation of the Pause Menu

4.6.3 Dead menu

The dead menu HUD Appears When defeated, the same interface as menu interfaces.

Figure 4.22: Implementation of the Dead Menu

4.6.4 Win menu

The winning menu HUD Appears When one winning the game:

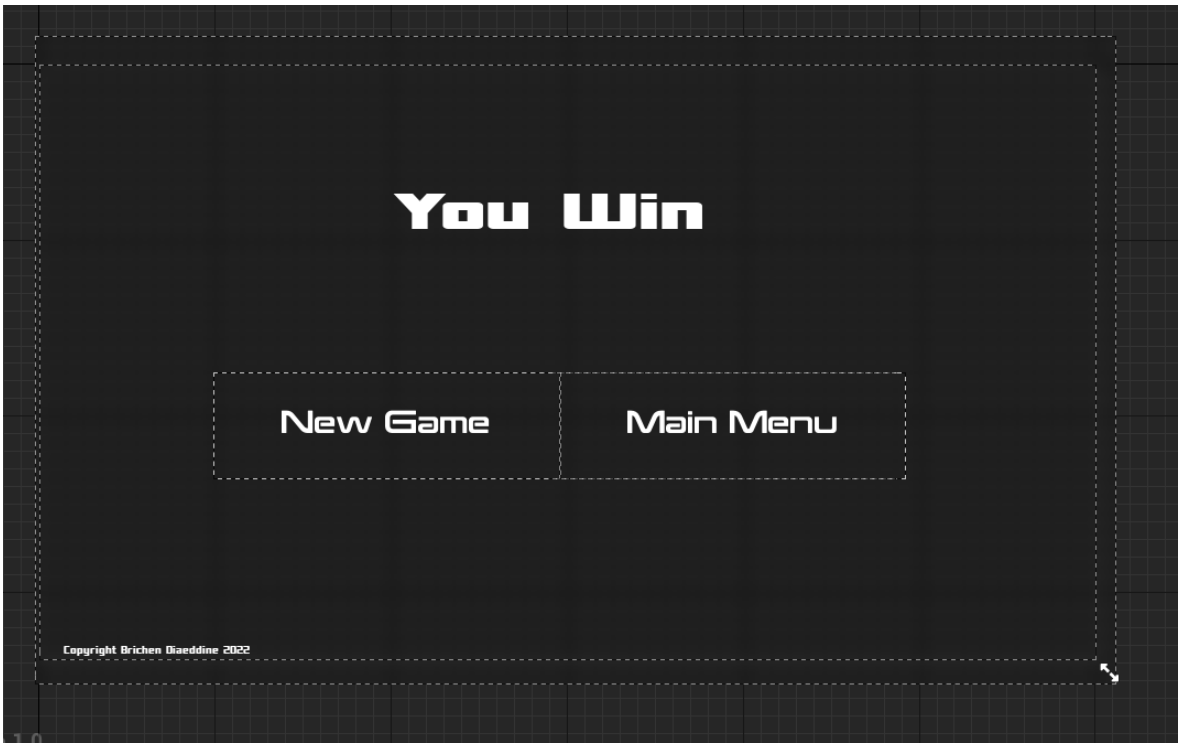


Figure 4.23: Implementation of the Win Menu

4.6.5 Main character HUD

The main character HUD contains a health bar composed of a progress bar, that is updated when the player character receives damage. Plus a Coin counter in the bottom left and an ammo counter in the bottom right.

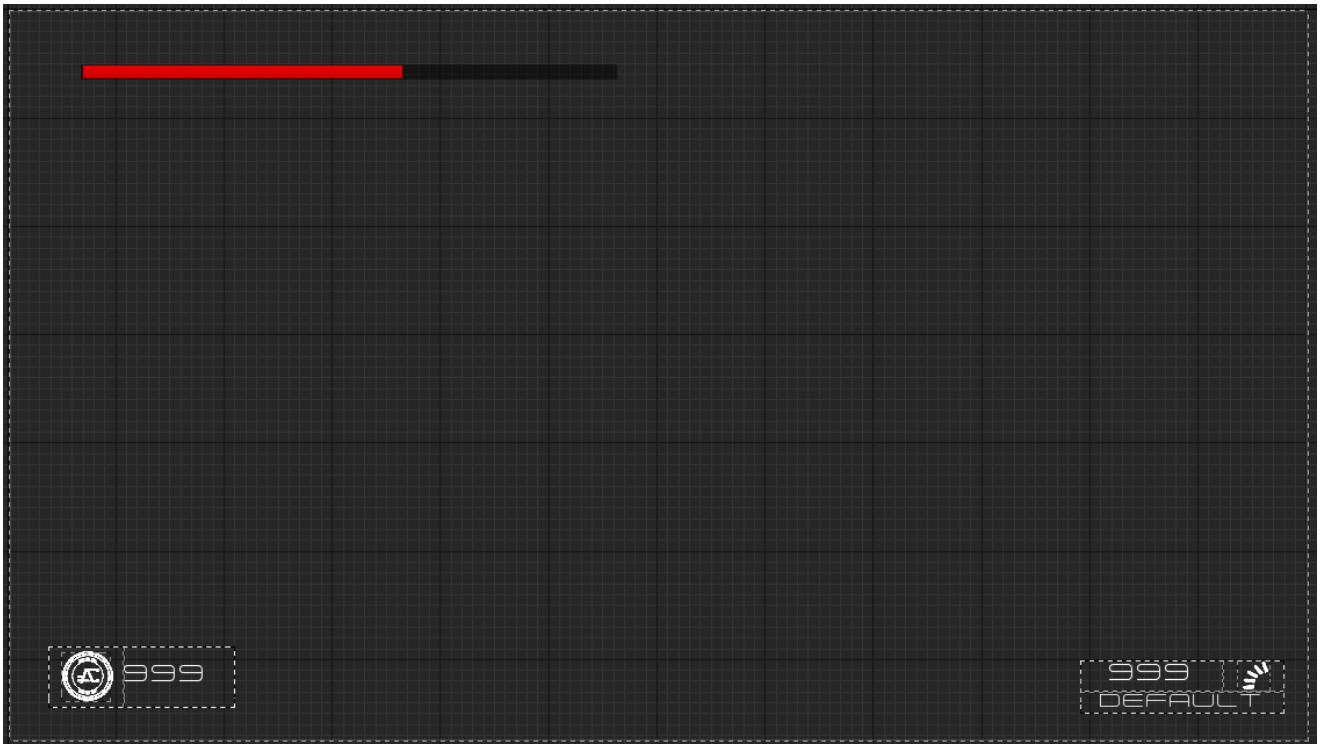


Figure 4.24: Implementation of the Main Character UI (HUD)

Figure 4.25: The main character UI in real time gameplay

4.6.6 Weapon HUD

The Weapon HUD is a simple widget, that shows when the main character interacts with the weapons that contain information about the weapon like name and the type.

Figure 4.26: Implementation of the weapons Widget UI

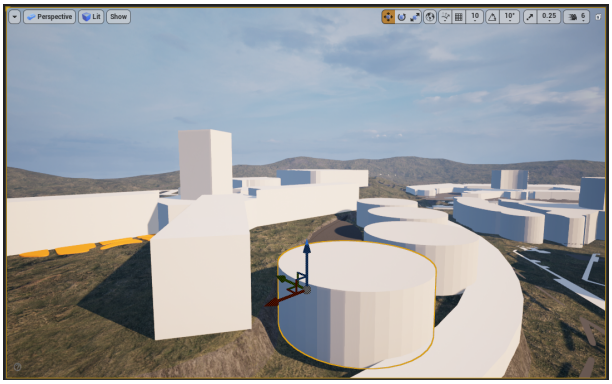
4.7 Implementation of other features to obtain the final game

Programming can be considered the most important part in the view of the game developer, but the consumer's view of the game is dominated by the external appearance of the game, so this aspect must be taken care of.

4.7.1 3D modeling

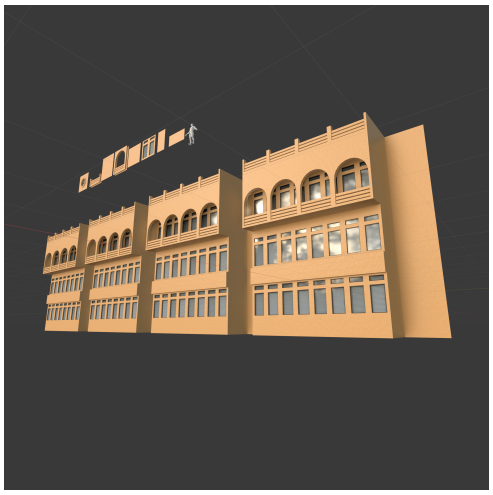
4.7.1.1 University model

Since the project is a graduation project, it was necessary to add a touch to the university, so We designed the university in realistic details, especially the part in which We studied :

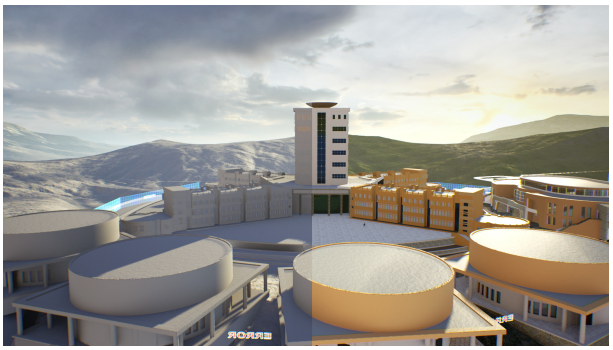


(a) prototyping the university

(b) Model parts design



(c) early stage of design



(d) Late stage of design

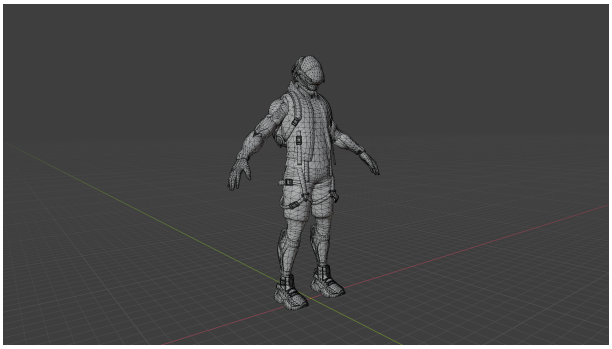
Figure 4.27: Overview of the university design stages

Figure 4.28: The university level design

Figure 4.29: The university level design (side part)

4.7.1.2 Characters models

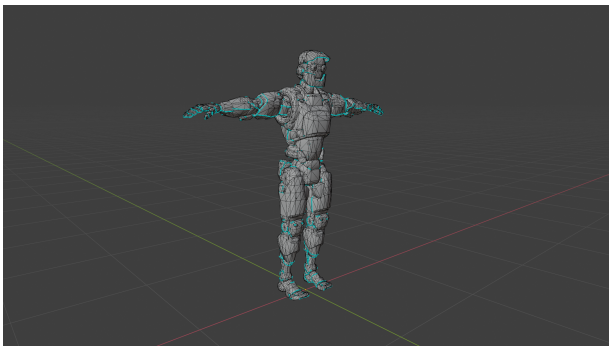
These are some of the modeled characters:



(a) 3d design in blender

(b) final result in unreal engine 4

Figure 4.30: The Main character



(a) 3d design in blender

(b) final result in unreal engine 4

Figure 4.31: The enemy character

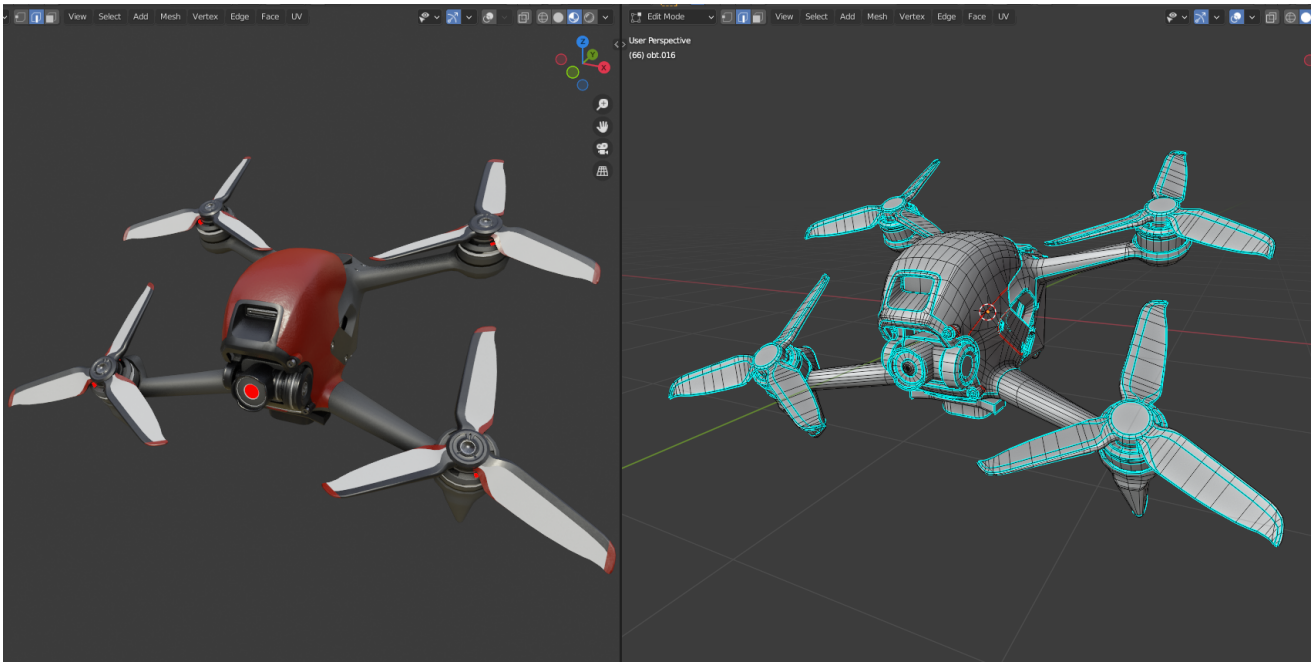


Figure 4.32: The Drone character

Figure 4.33: game design model

4.7.2 Animation

These are some of animations:

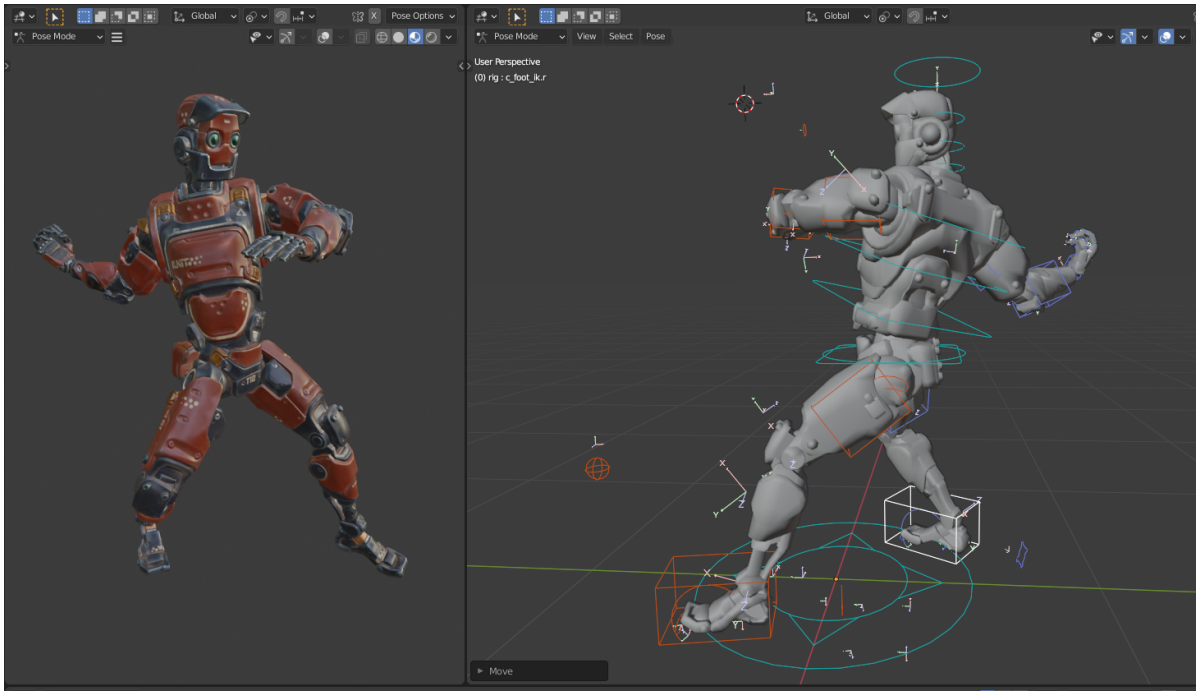


Figure 4.34: Character animation

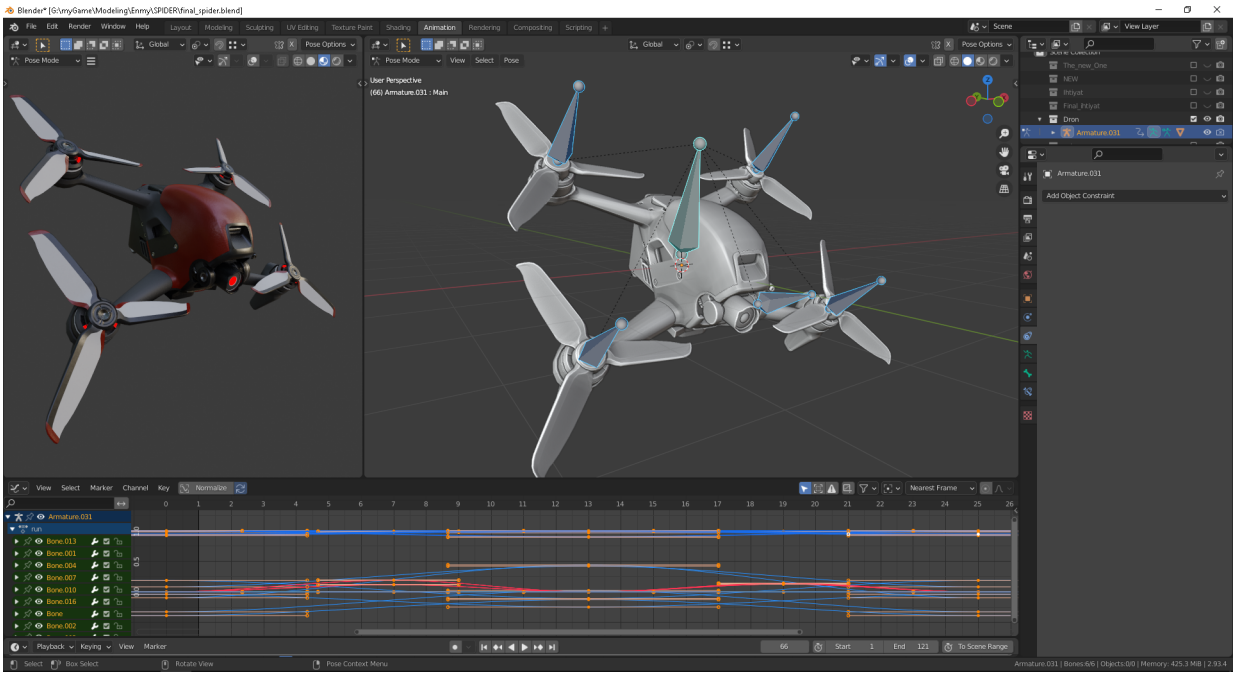


Figure 4.35: Drone animation

Figure 4.36: Project X

4.8 Conclusion

In this chapter, We have illustrated the details of the implementation of our multi-agent game system, We have cited the development tools that we used for the realization of our work. Then, We presented different parts of the system with detailed explanations of functionalities in the system.

General conclusion

Conclusion :
The objective of this project was to develop a game with an artificial intelligence-based multi-agent system that would model the behavior of enemies, and we have come close to achieving this goal. All the tasks to be accomplished, Almost completed and within the expected time frame, despite the difficulties encountered during the project completion stages. The motivation of the project's developer was to learn how to design and implement behavior trees to make non-player characters make decisions and execute actions inconsistent and realistic ways, as well as gain experience in game development and particularly in the implementation of game-oriented AI solutions, these objectives almost accomplished.

Behavior trees proved to be very flexible during the development process: it was easy to modify the structure of a tree to incorporate new character abilities and types of decisions. The final game's agents can detect characters and objects with vision and hearing. The perceived stimuli used to obtain information that can potentially help them achieve their goals by taking decisions and sharing their findings with other agents, who compare the received information with their knowledge. In other games, however, the reasoning capabilities of non-player characters are much more elaborate than in this game project, and agents participate in complex cooperative strategies.

The interaction between the agents in the developed MAS is not very complex (it was not necessary for the enemies' objective of chasing and defeating the player), while other games, such as some strategy games, have multi-agent systems in which agents interact in many creative ways. Some games include non-player characters that instead of trying to defeat the player has the objective of helping them to achieve their goals. It would be interesting to model the way how these characters try to interpret the player's intentions (observing the player character's actions) and suggest ways of achieving the shared goals: it would require defining a human-machine communication system.

The gameplay of this game includes some interesting player abilities, such as carrying weapons, and each weapon has a special fighting method, Adding health using health items, and winning coins, In addition to a movement system that is considered complex and scalable, in additional Fun fighting system and realistic graphics. This project was focused on the design of visuals since a part of the university was designed and made this design as a level in the game, taking into account the minute details of the university,

despite the weak capabilities of the device used.In addition, this is our first attempt at creating a system of this complexity and magnitude.

Bibliography

[1] Rabin. Introduction to Game Development (2nd ed., Vol. 1). Course Technology Cengage Learning.

[2] Rido Ramadan, Yani Widyani . Game development life cycle guidelines, Conference: 2013 International Conference on Advanced Computer Science and Information Systems (ICACSIS),DOI:10.1109/ICACSIS.2013.6761558

[3] Novak, J. (2011). Game Development Essentials: An Introduction (3rd ed., Vol. 1). Cengage Learning.

[4] Ralph Edwards, "The Game Production Pipeline: Concept to Completion". [online] (Jun 17, 2012) Available on : https://www.ign.com/articles/2006/03/16/the-game-production-pipeline-concept-to-completion#! (Consulted the 23 january 2022).

[5] TIMMER, JUDITH, et al. "GAMES ARISING FROM INFINITE PRODUCTION SITUATIONS." International Game Theory Review, vol. 02, no. 01, 2000, pp. 97–105. Crossref, https://doi.org/10.1142/s0219198900000020.

[6] Chandler, H. M. (2013). The Game Production Handbook (3rd ed., Vol. 1). Jones & Bartlett Publishers.

[7] by Eric Freedman,Engineering Queerness in the Game Development Pipeline,the international journal of computer game research,volume 18 issue 3 December 2018 ISSN:1604-7982

[8] Aleem, S., Capretz, L.F., Ahmed, F. Game development software engineering process life cycle: a systematic review. J Softw Eng Res Dev 4, 6 (2016).

[9] Devin Pickell, "The 7 Stages of Game Development". [online] (Apr 8, 2019) Available on :https://www.g2.com/articles/stages-of-game-development (Consulted the 23 january 2022).

[10] Saiqa Aleem, Luiz Fernando Capretz , Faheem Ahmed.Critical success factors to improve the game development process from a developer's perspective ,Journal of Computer Science and Technology, 31(5):925-950, DOI: 10.007/s11390-016-1673-z, Springer, September 2016.

[11] Nadia Stefyn, "How video games are made: the game development process". [online] (05/09/2022) Available on :https://www.cgspectrum.com/blog/game-development-process (Consulted the 23 january 2022).

[12] Scott Slatton, "Game Development Pipeline and Technologies". [online] (Apr 8, 2019) Available on :https://dev.to/scottslatton/game-development-pipeline-and-technologies-h0b (Consulted the 23 january 2022).

[13] Dawson, Chad, "Formations," AI Game Programming Wisdom, Charles River Media, 2002.

[14] Fu, Dan, and Houlette, Ryan, "Constructing a Decision Tree Based on Past Experience," AI Game Programming Wisdom 2, Charles River Media, 2003.

[15] Hu, J.; Bhowmick, P.; Jang, I.; Arvin, F.; Lanzon, A., "A Decentralized Cluster Formation Containment Framework for Multirobot Systems" IEEE Transactions on Robotics, 2021.

[16] Hu, J.; Turgut, A.; Lennox, B.; Arvin, F., "Robust Formation Coordination of Robot Swarms with Nonlinear Dynamics and Unknown Disturbances: Design and Experiments" IEEE Transactions on Circuits and Systems II: Express Briefs, 2021.

[17] Hu, J.; Bhowmick, P.; Lanzon, A., "Group Coordinated Control of Networked Mobile Robots with Applications to Object Transportation" IEEE Transactions on Vehicular Technology, 2021.

[18] Wiering, M. A. (2000). "Multi-agent reinforcement learning for traffic light control". Machine Learning: Proceedings of the Seventeenth International Conference (Icml'2000): 1151–1158. hdl:1874/20827.

[19] K.P., Sycara, Multiagent Systems, AI Magazine, vol. 19, no. 2, pp. 79-92, 1998.

[20] Gillies, M. (2009). Learning Finite-State Machine Controllers From Motion Capture Data. IEEE Transactions on Computational Intelligence and AI in Games.

[21] C. Castelfranchi and R. Conte. Understanding the effects of norms in social groups through simulation. In G. N. Gilbert and R. Conte, editors, Artificial Societies: the computer simulation of social life, pages 252-267. UCL Press, London, 1995.

[22] R. Conte, N. Gilbert, and J. S. Sichman. Mas and social simulation: A suitable commitment. In J. Sichman, R. Conte, and N. Gilbert, editors, International Workshop on Multi-Agent Based Simulation - MABS, volume 1534 of Lecture Notes in Artificial Intelligence, pages 1-9, Berlin, 1998. Springer - Verlag.

[23] E. Hurwitz and T. Marwala, "Learning to bluff: A multi-agent approach", IEEE International Conference on Systems, Man and Cybernetics, Montreal, Canada, (accepted)

[24] Brooks, Rodney, "How to Build Complete Creatures Rather than Isolated Cognitive Simulators," Architectures for Intelligence, Lawrence Erlbaum Associates, Fall 1989.

[25] Isla, Damian, "Handling Complexity in the Halo 2 AI," Game Developer Conference 2005.

[26] Isla, Damian, and Blumberg, Bruce, "Blackboard Architectures," AI Game Programming Wisdom, Charles River Media, 2002.

[27] Orkin, Jeff, "Simple Techniques for Coordinated Behavior," AI Game Programming Wisdom 2, Charles River Media, 2003.

[28] Cain02l Cain, Timothy, "Practical Optimizations for A* Path Generation," AI Game Programming Wisdom, Charles River Media, 2002.

[29] Higgins, Dan, "How to Achieve Lightning-Fast A*," AI Game Programming Wisdom, Charles River Media, 2002.

[30] Rabin, Steve, "A* Speed Optimizations," Game Programming Gems, Charles River Media, 2000.

[31] Schmitt, J., & Kostler, H. (2016). A multi-objective genetic algorithm for simulating optimal fights in StarCraft II. 2016 IEEE Conference on Computational Intelligence and Games (CIG).

[32] Sweetser, Penny, "How to Build Evolutionary Algorithms for Games," AI Game Programming Wisdom 2, Charles River Media, 2003.

[33] OrkinRabin, Steve, "Filtered Randomness for AI Decisions and Game Logic," AI Game Programming Wisdom 2, Charles River Media, 2003.

[34] Reynolds, Craig, "Flocks, Herds, and Schools: A Distributed Behavioral Model," Computer Graphics, 21(4) (SIGGRAPH '87 Conference Proceedings).

[35] Sebastian Castro, "Introduction to behavior trees" [online] (August 17, 2021) Available on :https://robohub.org/introduction-to-behavior-trees/ (Consulted the 10 April 2022).

[36] Bazzan, A. L. C. (2012). Coordinating many agents in stochastic games. The 2012 International Joint Conference on Neural Networks (IJCNN).

[37] Weibing, L., Xianjia, W., & Binbin, H. (2009). Evolutionary Markov Games Based on Neural Network. Lecture Notes in Computer Science.

[38] Laslier, J.-F., & Walliser, B. (2005). A reinforcement learning process in extensive form games. International Journal of Game Theory.

[39] Alt, Greg, and King, Kristin, "A Dynamic Reputation System Based on Event Knowledge," AI Game Programming Wisdom, Charles River Media, 2002.

[40] Brockington, Mark, "Building A Reputation System: Hatred, Forgiveness, and Surrender in Neverwinter Nights," Massively Multiplayer Game Development, Charles River Media, 2003.

[41] Cain, Timothy, "Practical Optimizations for A* Path Generation," AI Game Programming Wisdom, Charles River Media, 2002.

[42] Evans, Richard, "Varieties ofLearning," AI Game Programming Wisdom, Charles River Media, 2002.

[43] Gibson, James, The Ecological Approach to Visual Perception, Lawrence Erlbaum Assoc, 1987.

[44] Grimani, Mario, "Wall Building for RTS Games," AI Game Programming Wisdom 2, Charles River Media, 2003.

[45] Hargrove, Chris, "Simplified Animation Selection," AI Game Programming Wisdom 2, Charles River Media, 2003.

[46] Hargrove, Chris, "Pluggable Animations," AI Game Programming Wisdom 2, Charles River Media, 2003.

[47] Ji, L. X.,& Ma, J. H. (2014). Research on the Behavior of Intelligent Role in Computer Games Based on Behavior Tree. Applied Mechanics and Materials.

[48] Tozour, Paul, "Search Space Representations," AI Game Programming Wisdom 2, Charles River Media, 2003.

[49] Elhadi Shakshukia, Malcolm Reidb ,Multi-Agent System Applications in Healthcare: Current Technology and Future Roadmap - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/figure/An-example-of-a-Multi-Agent-System-for-decision-support-6_fig1_277727183 (accessed 20 May, 2022)

[50] Fatih Küçükkarakurt, Game Programming Fundamentals. [online] (August 25, 2021) Available on : https://www.developer.com/languages/game-programming-fundamentals/ (Consulted the 20 April 2022).

[51] Kat Boogaard, What is a concept map. [online] (March 29, 2021) Available on : https://miro.com/blog/what-is-concept-map/ (Consulted the 20 April 2022).

[52] Pluralsight , Designing a HUD That Works for Your Game. [online] (March 5, 2014) Available on : https://www.pluralsight.com/blog/film-games/designing-a-hud-that-works-for-your-game?exp=2 (Consulted the 20 April 2022).

[53] Sapio, F. (2019). Hands-on artificial intelligence with Unreal Engine: Everything you want to know about game Ai using blueprints or C++. Packt.

[54] Reinhold Preiner, Content Creation for a 3D Game with Maya and Unity 3D - Scientific Figure on ResearchGate. Available from: https://www.researchgate.net/publication/267417785_Content_Creation_for_a_3D_Game_with_Maya_and_Unity_3D (accessed 5 May, 2022).

[55] the MasterClass staff ,Gaming 101: Guide to Video Game Programming Languages. [online] (Nov 8, 2020) Available on : https://www.masterclass.com/articles/guide-to-video-game-programming-languages#7-video-game-programming-languages (Consulted the 20 Mai 2022).

[56] Fatih Küçükkarakurt ,Game Programming Fundamentalss. [online] (August 25, 2021) Available on : https://www.developer.com/languages/game-programming-fundamentals/ (Consulted the 20 Mai 2022).

[57] Fatih Küçükkarakurt ,Sourcetree A GUI for Git and Mercurial that is easy for beginners and powerful for experts. [online] Available on : https://www.globallogic.com/services/offerings/atlassian/products/sourcetree/ (Consulted the 20 Mai 2022).

[58] blender team ,about blender. [online] Available on : https://www.blender.org/about/ (Consulted the 20 Mai 2022).

[59] Wayne Maxwell ,What is Quixel Mixer and Why You Should Give it a Go. [online] (October 9, 2021) Available on : https://www.developer.com/languages/game-programming-fundamentals/ (Consulted the 20 Mai 2022).

[60] Computer Hope ,Visual Studio. [online] (06/07/2019) Available on : https://www.computerhope.com/jargon/v/visual-studio.htm (Consulted the 20 Mai 2022).

[61] UE Team ,Unreal Engine 5 Documentation. [online] Available on : https://docs.unrealengine.com/5.0/en-US/ (Consulted the 20 Mai 2022).

[62] UE Team ,Environment Query System. [online] Available on : https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/ArtificialIntelligence/EQS/ (Consulted the 20 Mai 2022).