Real-Time Automated Planner in Video Games

Author: Margaux LuSSIANA
Student number: H00281112

Supervisor: Dr. Stefano PADILLA
Second reader: Dr. Santiago CHUMBE

A thesis submitted in fulfillment of the requirements
for the degree of MSc in Artificial Intelligence with SMI
in the
School of Mathematical and Computer Sciences

August 2018
Declaration of Authorship

I, Margaux LUSSIANA, declare that this thesis titled, 'Real-Time Automated Planner in Video Games' and the work presented in it is my own. I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed: 

Date: 13/08/2018
“The obvious objective of video games is to entertain people by surprising them with new experiences.”

Shigeru Miyamoto
Abstract

This report seeks to present a project aiming at producing a real-time planning system able to provide clever reactions to world changes in a video game environment. The two hypotheses of the project draw attention around the real-time aspect of the planner and the impact of artificial intelligence in video games. The prototyping of the automated planner and its implementation inside a game environment help providing results and analyses to the testing experiments in order to validate the planner architecture, performances and methods.
Acknowledgements

The acknowledgements go to my supervisor Dr. Stefano Padilla who gave me a great freedom for this project and supported my choices.

I also would like to thank Mr. Jeff Orkin, a renowned developer for his work in AI in games, who accepted to answer some of my questions and put me in contact with Mr. Eric Jacopin for further explanations and helps about planners.

Finally, I wish to thank Mr. Eric Jacopin who accepted to help me during my project execution, providing me advices to carry out the project in good conditions.
# Contents

Declaration of Authorship  i

Abstract  iii

Acknowledgements  iv

Contents  v

List of Figures  viii

Abbreviations  ix

1 Presentation of the Project  1
  1.1 Introduction  1
  1.2 Description of the project  2
    1.2.1 Aim  2
    1.2.2 Objectives  3
    1.2.3 Personal Motivation  4
  1.3 In Closing  4

2 Literature Review  5
  2.1 Planners Characteristics  5
    2.1.1 Decision-making  6
    2.1.2 Planning  6
    2.1.3 Search algorithms  7
    2.1.4 Real-Time vs Off-Line Planners  8
    2.1.5 Languages  9
    2.1.6 Tools  10
  2.2 Useful works  10
    2.2.1 Planning systems in games  11
    2.2.2 Reference 1 : Agent Architecture Considerations for Real-Time Planning in Games by Jeff Orkin  11
    2.2.3 Reference 2 : Applying Goal-Oriented Action Planning to Games by Jeff Orkin  12
2.2.4 Reference 3: A Real-Time PDDL-Based Planning Component for Video Games by Eric Jacopin ........................................... 12
2.2.5 Talks with Mr Jacopin ............................................. 12
2.3 Unity Engine ......................................................... 13
  2.3.1 Unity terminologies ........................................... 13
  2.3.2 Unity tools ..................................................... 14
2.4 In Closing ......................................................... 14

3 Ethics and Risks ...................................................... 15
  3.1 Ethical Approval .................................................. 15
  3.2 Risks Assessment ................................................ 15
  3.3 In Closing ....................................................... 16

4 Development of the Project .......................................... 17
  4.1 Steps ............................................................... 17
    4.1.1 Development ............................................... 17
    4.1.2 Evaluation ................................................ 18
  4.2 Structure ......................................................... 18
    4.2.1 Planner and Language .................................... 18
    4.2.2 Applications and Tools .................................. 19
    4.2.3 Scripts .................................................... 19
  4.3 Game environment ............................................. 20
    4.3.1 Game properties and mechanics ......................... 20
    4.3.2 Visual features ......................................... 22
    4.3.3 Game Characters ....................................... 23
      4.3.3.1 Creation of the Artificial Agent’s tasks ............. 24
  4.4 Creation of the Planner ........................................ 26
    4.4.1 Knowledge Representation ................................ 26
    4.4.2 A* Algorithm ............................................ 27
    4.4.3 Best-First Algorithm ................................... 28
    4.4.4 Implementation of the AI ................................. 29
    4.4.5 A temporary testing and debugging environment ......... 29
    4.4.6 Adaptation with the actual game ....................... 30
  4.5 In Closing ..................................................... 30

5 Test and Evaluation .................................................. 32
  5.1 Quantitative testing ........................................... 32
    5.1.1 Conditions of testing .................................... 33
    5.1.2 Planners ................................................... 33
    5.1.3 Heuristics ................................................. 34
    5.1.4 Comparisons and statistics ............................... 34
  5.2 Qualitative testing ............................................ 35
    5.2.1 Progress ................................................... 35
    5.2.2 Questionnaire ........................................... 35
    5.2.3 Theme Analysis .......................................... 36
  5.3 In Closing .................................................... 36

6 Results ............................................................. 37
## Contents

6.1 Quantitative Results ................................................. 37  
6.1.1 T-Tests ...................................................... 37  
6.1.2 Real-Time analysis ........................................... 39  
6.1.3 Subjective quality of plans and integration analyses ........... 40  
6.2 Qualitative Results ............................................... 41  
6.2.1 Theme analysis ............................................... 41  
6.2.2 Scores ....................................................... 42  
6.2.3 Other comments .............................................. 43  
6.3 Achievements ..................................................... 43  
6.4 Conclusions ....................................................... 44  

7 Discussion .............................................................. 46  
7.1 Project difficulties ................................................ 46  
7.2 Future Work ....................................................... 47  
7.3 Self-Actualization ................................................. 47  
7.4 AI in games ......................................................... 47  
7.5 Conclusion ........................................................ 48  

A NeoPacman, a student project in Python .............................. 49  
B Coursework – Search, Knowledge Representation, and Planning ....... 51  
C Risks Assessment .................................................... 53  
D Search Algorithms .................................................. 54  
E Qualitative Evaluation - Questionnaire ................................ 56  

Bibliography .................................................................. 60
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Game platform</td>
<td>20</td>
</tr>
<tr>
<td>4.2</td>
<td>Fire Action</td>
<td>21</td>
</tr>
<tr>
<td>4.3</td>
<td>Collision <em>OnTriggerEnter</em></td>
<td>22</td>
</tr>
<tr>
<td>4.4</td>
<td>HUD</td>
<td>23</td>
</tr>
<tr>
<td>4.5</td>
<td>Agent actions scene</td>
<td>25</td>
</tr>
<tr>
<td>4.6</td>
<td>Patrol <em>Coroutines</em></td>
<td>26</td>
</tr>
<tr>
<td>4.7</td>
<td>Pop <em>Custom Method</em></td>
<td>27</td>
</tr>
<tr>
<td>4.8</td>
<td>List of solutions</td>
<td>28</td>
</tr>
<tr>
<td>4.9</td>
<td>Communication methods</td>
<td>29</td>
</tr>
<tr>
<td>4.10</td>
<td>Tracking Ammo System <em>Coroutine</em></td>
<td>30</td>
</tr>
<tr>
<td>6.1</td>
<td>Planning Execution Time Values, BFS</td>
<td>38</td>
</tr>
<tr>
<td>6.2</td>
<td>Planning Execution Time Values, A*</td>
<td>38</td>
</tr>
<tr>
<td>6.3</td>
<td>Frames per Seconds, Table</td>
<td>40</td>
</tr>
<tr>
<td>6.4</td>
<td>Scores, Evaluation</td>
<td>43</td>
</tr>
<tr>
<td>A.1</td>
<td>NeoPacman Code</td>
<td>49</td>
</tr>
<tr>
<td>A.2</td>
<td>NeoPacman screen</td>
<td>50</td>
</tr>
<tr>
<td>B.1</td>
<td>Action in Domain File</td>
<td>52</td>
</tr>
<tr>
<td>B.2</td>
<td>Goal and Plan</td>
<td>52</td>
</tr>
<tr>
<td>C.1</td>
<td>Risks Assessment</td>
<td>53</td>
</tr>
<tr>
<td>D.1</td>
<td>A* Algorithm</td>
<td>54</td>
</tr>
<tr>
<td>D.2</td>
<td>Best-First Search Algorithm</td>
<td>55</td>
</tr>
<tr>
<td>E.1</td>
<td>Questionnaire, p.1</td>
<td>57</td>
</tr>
<tr>
<td>E.2</td>
<td>Questionnaire, p.2</td>
<td>58</td>
</tr>
<tr>
<td>E.3</td>
<td>Questionnaire, p.3</td>
<td>59</td>
</tr>
</tbody>
</table>
Abbreviations

AI  Artificial Intelligence
AA  Artificial Agents
HTN Hierarchical Task Network
PDDL Planning Domain Description Language
GOAP Goal Oriented Action Planning
BDI Belief-Desire-Intention
FSM Finite State Machine
BFS Best First Search
AS A* Search
KR Knowledge Representation
FPS First Person Shooter
RTS Real-Time Strategy
MOBA Multiplayer Online Battle Arena
NPC Non Playing Character
To my inner geek, and my wish for an amazing meeting between awesome artificial intelligent technologies and immersive video games...
Chapter 1

Presentation of the Project

The project’s name “Real-Time Automated Planner in Video Games” covers the main goal of it. By splitting this title, we can understand the domains involved in this project. Automated planning is a technology of Artificial Intelligence, that still needs to be improved and have potential in many areas. Video games state the environment where the planner will be developed, as it allows us to disconnect from the reality and its constraints. Finally, the real-time aspect is probably the most important one as it states what I am aiming at: allowing the planner to adapt itself promptly whilst facing a changing situation. This technology brings great possibilities to video games, but could also be widen to other domains, such as home automation.

1.1 Introduction

The game industry always needs to improve the player experience either by stunning graphics or more immersible game-plays. The requirements for immersible games expanded with years and game developers require to meet the expectations of players. One of the most requested and widespread aspect is the use of Artificial Intelligence and Artificial Agents that act and react to the game changes, in what seems to be an intelligent way. This technology can lead the way to many possibilities in future games and might revolutionize game-plays for many game styles as it evolves.

Video games with AI and Intelligent Agents are already numerous. There are many ways of implementing them depending on the degree of smartness illusion that is needed. Basic troopers can have a simple path-finding algorithm to simulate soldiers, but space-ships may need a more complex intelligence to handle destinations and occupants. Even if Artificial Intelligence in video games is more ”artificial” than ”intelligent”, it has already
managed to make its way to games, such as in FEAR by Monolith Studio, and be almost irreplaceable.

AI in games can take many forms, but automated planning is the one that interest me the most. I believe that planning in Real-Time could become the next standard for Artificial Intelligence in games. The real-time aspect can be usable in on-line games for a faithful response to world changes. However, any video games could benefit from it as well. Right now, most of AI in games are hard coded and require a human operator to create all or part of the AI reactions. But with automated planners developers would not worry about the adaptability of the agent according to world changes. The agent itself can self-select actions and create its own plan, instead of searching in a pre-made list of possible plans that developers would have think of. The master project I am engaged into aims at creating a real-time planner capable of great performances and re-usability without having to anticipate possible plans.

1.2 Description of the project

The scope of this project is about dabbling the numerous opportunities of a growing concept of AI: automated planners. This 10 year-old area of AI aims at replicating human abilities for making complex actions and taking decisions in multi-dimension worlds. In that sense, it fits perfectly the context and problems of video games and gives an accurate answer to the need of AI.

Automated planning is a broad notion and it gathers several technologies. They are centered around actions and plans, but their minor differences give them various usages depending on needs and situations. To briefly name some, HTN Based planners or GOAP planners are often used in games. It can be very difficult to pretend that an agent is smart especially when this agent interacts directly with humans, such as in video games. The capacities of the agent need to meet the player expectations, and those different types of planners offer different ways of achieving this by simulating human intelligence and behavior, in accordance with problem definition and world simulation.

1.2.1 Aim

The goal of this project is not only to prototype an intelligent agent but also to dupe the user as much as possible with this intelligence by achieving a human-like behavior. This particular AI doesn’t need to expressly pass the Turing test, but the user need to believe in it. It should be difficult, nay impossible, to determine either it is a machine or a
human playing. To this is added the specificity that the planner simulating intelligence should be real-time adaptable to better fit strategy or FPS games. I would like to add that real-time adaptability is real important for me as lots of games are decisions dependent; they have several options and opportunities depending on the player own choices and decisions during the game. I strongly believe that the real-time aspect of AI could become a main feature in future video games.

My two main hypothesis questions for this project are:

1. Is planning a suitable AI technology for real-time video games in terms of performance?

2. How are intelligent agent impacting players involvement and fun in video games?

The literature review further developed shows interesting usages of planning in video games. My goal is not to dispute the fact that planning is already used in video games and suits their requirements, but to convince about the great value that it adduces.

1.2.2 Objectives

The main objectives I want to achieve in this project are:

- Properly build a planner that suits the video game requirements
- Using the planner to give an intelligent behavior to an AA
- Evaluate the capacities of Planner technologies in real-time games
- Measure the impact of the AI on players involvement in games

The first 2 objectives are coding objectives. I want to create the program on my own and go further in planner technologies than I had during my previous lessons. The 3rd objective aims at evaluate my work but also investigate why and how my planner can be efficient, what could be its strengths and weaknesses; is the AI suitable from a performances point of view? The last objective is a more social objective that aims at gathering feedbacks in order to give sens to my work; is the AI suitable from the player point of view?
1.2.3 Personal Motivation

This project is a personal challenge to myself in terms of technology and knowledge. My previous lesson on AI offered me a large view over the different AI technologies, but my will is to go deeper in understanding and creating such AI. I chose a video game environment to develop my system as I think AI and games match greatly in terms of requirements. Games allow almost any sort of ideas and AI can be easily tested in games.

1.3 In Closing

This chapter referred to my personal decisions about this project, how I imagine it to be done and what are my motivations. The next chapter will discuss about the current state of art of AI in games, especially about planners.

Report written by —
Margaux Lussiana
Chapter 2

Literature Review

AI can have many forms and uses depending on cases. For instance, my previous work presented in Appendix A shows a very simple design of AI using if statements to create illusion of simple but smart behaviors, figure A.1. However, the wanted technologies for the project need not only to illusion smartness but also to illusion reflexion and problems resolution. This is why planners are discussed in this chapter, as they fulfill all the requirements to help me achieving my project’s goals.

There are several points that distinct planners from each others, or from other AI technologies: reasoning methods, knowledge base, planning approach. By combining them, one can construct a specific planner that best fit its requirements. For the scope of this project, we will discuss about existing planners that are made up of a joint of those characteristics and that could help me to produce an efficient planning system for games. The presented planners could be inspirational systems to support my thoughts and strategies.

2.1 Planners Characteristics

There are two main types of design for planners. One is based on Final State Machines while the other rely on search methods. FSM planners are design for decision-making, using pre-made plans. Search planners work with search algorithms and can, along with making decisions, proceed to plan.
Chapter 2. Literature Review

2.1.1 Decision-making

FSMs allow game developers to implement decision-making AI. Finite State Machine is the first technology developers are trending towards for a simple implementation of AI without a complex code. FSM is a model used to represent world states, and provides great results in small and simple environment. However, when problems and worlds get larger, it requires more coding resources as FSM states need to be anticipated by developers.

Hierarchical Task Network is a second model of decision-making planner. HTN planners such as SHOP2 Au et al. [2011] resemble FSM planners where tasks are decomposed into primitives. A primitive action is the lowest level of action possible, in terms of conditions and effects, that can be conduct by the agent. A task can be define by a complex arrangement of simple actions leading to the goal. However, HTN are top-down approaches to planning where developers pre-define how goals can be subdivided. It means HTN evaluate all possible plans, select the most likely to fulfill the goal and decompose it in several simpler actions. Hence, HTN requires the developers to anticipate possible plans ahead of time, just like FSM.

Belief-Desire-Intention is another type of decision-making planner. BDI is a model that relies on selection of plans among a pre-defined plans list and executing those plans according to goals constraints and conditions. They are very similar to HTN, as they do not create plans but make a decision among states. However, BDI planners can assemble and execute plans whilst proceeding to the solution finding process, meaning that BDI can be used in real-time environments. BDI and HTN systems are often used similarly as they can produce equivalent results depending on the implementation method chosen by developers, as stated by De Silva and Padgham [2004].

Planners relying on FSM-like methods need big coding-resources to find a path to any complex goal, as actions and conditions should be stated by developers beforehand. The planner go through the possible plans and select one of them to fulfill the goal, often being an optimum plan for a given state. This architecture allows the planner to make decisions on which action, or set of actions, to do in order to achieve the goal. The planner only has to select actions using preconditions constraints, without any sort of planning concepts or reasoning methods towards effects apply on the state space.

2.1.2 Planning

Search planners are designed to provide decision-making and planning capabilities. Along with selecting actions, they can create plans by choosing how to perform those
actions. They self-manage the path through the goal.

STRIPS is a classical planning representation underlying the basis of several other planning systems and automated planners. To be brief, STRIPS makes simple, and not always realistic, assumptions such as closed world and complete world states to bring satisfying solutions of planning.

Another newer planner, that I found more interesting, is the GOAP planner that has already been used in video games such as *F.E.A.R.* by Monolith Productions\(^1\). GOAP is a simplified version of STRIPS. By pre-defining a data structure for the world state, GOAP can do world state updates and lookups very quickly. GOAP natively supports preconditions by querying the real game world, and without using the internal world state data structure of the planner. It can be useful for doing line of sight tests for instance, that cannot be easily represented by a discreet variable.

GOAP and STRIPS are bottom-up approaches to planning, where planners use small atomic building blocks (actions) to search for a valid plan, and build it, in order to satisfy some goal. Hence, GOAP and STRIPS can generate plans that the developer did not even conceive of.

Fast downward by Helmert [2011] is also a complete planner deliver with several configurations of running and search algorithms. This planner can be used with different features, allowing the user to select various things, such as running a plan validator or simply specify search options. This planner was partly used in my previous work presented in Appendix B, mainly as a search algorithm.

Those planners give a more intelligent and unpredictable behavior to agents compared to the traditional FSM ones. I will mainly focus on search planners, and thus search algorithms, to support this project.

### 2.1.3 Search algorithms

Automated Planners need algorithms to be processed. Planners contain the goal, the actions and the space state, but only the algorithm link them together in order to find a plan. There are numerous algorithms that can be used. During my previous work (Appendix B), I noticed that for a same planner, some algorithms were more efficient than others. Hence, algorithms are problem-dependent to find optimum plans. Efficiency of a planner takes into account process time but also quality of plans. I also noted that some algorithms were enable to find any plan, meaning that several algorithms may need to be tested on a planner in order to better choose one.

\(^1\)https://www.lith.com/games/fear
The most known search algorithm is A* search, which is also used in pathfinder mechanisms. It uses a heuristic that needs to be defined before running the planner to guide the search engine. The Lazy algorithm of Fast Downward settings also uses a heuristic search. However, even if heuristic search are common in planners, one can also find other search algorithms. Greedy-first search or best-first search are also used in pre-made settings of the Fast Downward planner. Those search algorithms and settings were used in the coursework presented in Appendix B and can be found on the Fast Downward website\(^2\); I will not describe all of them, as they are case-dependent.

### 2.1.4 Real-Time vs Off-Line Planners

For the scope of this project, the real-time aspect is very important. I aim at finding a planner configuration that would allow fast planning for real-time games. But real-time planning implies simple problems to solve, as the planner would need to perform re-planning steps often. Complex problems are resources-expensive, which is nearly impossible to provide in real-time games.

Several papers compare off-line and on-line planning in their analysis. The first one can provide complex plans and solve complex goals, as it disposes of all the time it requires to run. On the other hand, on-line planning need to be fast and only has short amount of time to perform planning. Thus, its goals need to be small and simple, and its plans will be short in terms of number of actions to be performed by the agent. On-line planners need to be able to quickly re-evaluate their plans and goals, and re-produce plans at any time and any world changes. A brief but clear comparison is made by Menif et al. [2014], while discussing about characteristics of a specific planner (SHPE).

There are two different planners I would like to introduce in this section that produced plans in real-time. The first one is the simpleFPS planner describes in Vassos and Papakonstantinou [2011]. simpleFPS is a STRIPS planner coded with PDDL that provides basic behaviors for NPCs in FPS games. It is supposed to be a first step toward the use of PDDL for AI implementation in games. simpleFPS allows the user to add some preferences in order to personalize the behaviors of the agents. This paper tries to highlight the success of PDDL and STRIPS planners in real-time games, more specifically in FPS games with fast world changes. simpleFPS was clearly designed to be a real-time planner, performing simple planning tasks in run-time making it usable in FPS games. The SHPE planner, Menif et al. [2014], based on the simpleFPS planner, also claims to perform planning in a very short run-time, allowing this planner to be real-time usable, even though it was not mainly design to be part of a real-time video game.

---

\(^2\)http://www.fast-downward.org/Doc/SearchEngine
As already stated in previous sections, BDI or HTN can also be used in real-time domains depending on their implementations, but those will not interest me for the scope of this project as they are designed for decision-making planners. However, GOAP planners are a great compromise between complex architectures such as simpleFPS planners and small range of actions. Those planners provide a great planning system and allow real-time usages, using simple goals and search algorithms.

2.1.5 Languages

Behavior trees, or other graphical languages, are easy to understand and manipulate for humans. For instance, 2APL language is a BDI-based language that allows the development and management of multiple agents. However, I will not use those languages for the project, as my agent stands alone and the coding part would be restricted. Instead, two options are available: planning languages or object-oriented languages.

The first ones are expressive languages allowing a great freedom and comprehension for actions declarations and conditions definitions. STRIPS is the oldest planning language created, used in many simple and old planners. It defines planners with two distinct files, domain and problem, and contains declarations of an initial state, some actions and a goal. This language stated the basis for another modern language called PDDL, and more recently PDDL 3.0. PDDL is used in the International Planning Competition Series\(^3\) and is regularly improved thanks to the competition results. PDDL3 is the most advanced version and allows more complexity in planning definitions. Some modules have been added to refine its available capabilities, such as the addition of constraints and preferences further explained in Gerevini and Long [2005] and Sohrabi et al. [2009].

However, the use of object-oriented languages can be appropriate when coding a planner for video games. Video games are mostly developed in object-oriented languages such as Python or C#, two powerful object-oriented languages that can handle a planning approach and even more help the embedding of such planners. The SHPE planner Menif et al. [2014] was developed with C++ language for two main reasons. "First, this is a very commonly used language for video games and we expect our choice to ease the integration of the planner in most game engines. Secondly, being a low-level compiled language, C++ seemed to be an appropriate option to achieve the best runtime" have stated the authors. The ease of integration of planners into games coding environment is important for developers and companies, as it could save time for the overall implementation of the game. The second point drew by the authors need further experimentations, especially

---

\(^3\)The International Planning Competition is organized in the context of the International Conference on Planning and Scheduling (ICAPS). Ninth edition in 2018: [https://ipc2018.bitbucket.io](https://ipc2018.bitbucket.io)
to compare several object-oriented and low-level compiled languages in term of runtime performance and its benefits in real-time planning and games.

2.1.6 Tools

Several tools and extension help developing planners. For instance, the Fast Downward pre-configurations\(^4\) offer a usable planner for an easy implementation or testing. Those configurations helped us a lot for a fast development of the space-ship planner (Appendix B).

PDDL studio by Plch et al. [2012] is a tool developed in order to simplify PDDL documents management. It allows the user to proceed with planner integration in command line system, along with other features that helps creating and maintaining PDDL projects. The entire software was design to support the user in PDDL coding, especially with syntax and semantic errors detection. Those features could be the most useful ones for my master project. As I already faced PDDL syntax and its difficulties, I believe PDDL studio can greatly help the comprehension and manipulation of PDDL specificities.

As previously stated, the implementation of planners using object-oriented languages is viable. Furthermore, a Python library\(^5\) helps managing PDDL files by providing a parsing tool and ease the implementation of PDDL planners in Python codes. A C++ library\(^6\) is also available, more specifically implemented for motion planners (pathfinding) using search based planning. Finally, a C# package\(^7\) shows that implementations of planners in such languages are grossly possible.

However, for the purpose of this project I aim at creating a planner and its settings on my own in order to specifically design it for the given case and not risk compatibility problems for re-adapting existing planners.

2.2 Useful works

During my literature review, there were some references that drew my attention. They could be a real inspiration for my work on this project.

---

\(^4\)Some configurations available in Examples section: \url{http://www.fast-downward.org/PlannerUsage}

\(^5\)pddlpy available at \url{https://pypi.python.org/pypi/pddlpy/0.1.9}

\(^6\)SBPL available at \url{https://github.com/sbpl/sbpl}

\(^7\)ReGoap, available at \url{https://github.com/luxkun/ReGoap}
2.2.1 Planning systems in games

This section briefly outlines three games that make a use of planners.

The first one, NO ONE LIVES FOREVER 2\(^8\) embeds plans and use FSM technology to manage artificial agents. This work will not truly influence my master project, but it proves that the use of FSM in games is a long time method and appears to be efficient.

The second and third ones, FEAR\(^9\) and FALLOUT 3\(^10\), make the use of GOAP systems improving the AI quality compared to FSM. FEAR is taken as example and support in the paper written by Orkin [2005] and shows the benefits of not only a decision-making system but also a real planning system in games. Further describing those games would not be relevant for the scope of this project, as I aim at developing a re-usable planner for any genres of games, along with a home-made game environment for testing purposes.

2.2.2 Reference 1 : Agent Architecture Considerations for Real-Time Planning in Games by Jeff Orkin

Jeff Orkin\(^11\) is an important player in AI development, especially in video games. He has worked for Monolith Productions\(^12\) and Warner Bros Games\(^13\) on several projects involving AI and planning. His works and papers can be found on his personal website. He has been a great source of inspiration for me while writing this paper. His work on F.E.A.R and NOLF2 contained useful information about the planning domain in games.

This paper Orkin [2005] written by Jeff Orkin explains the use of GOAP planners in games, especially in the F.E.A.R. game. Its most interesting aspect is the real-time implementation that fits perfectly my master project. He also wrote several other papers about AI in games and planning that I will definitely keep in mind as inspirations.

I contacted Mr Orkin during this report redaction and he kindly answers few questions to help me understanding this field.

\(^8\)https://www.lith.com/games/no-one-lives-forever-2
\(^9\)https://www.lith.com/games/fear
\(^10\)https://fallout.bethesda.net/games/fallout-3
\(^11\)http://alumni.media.mit.edu/~jorkin/
\(^12\)https://www.lith.com/
\(^13\)https://www.warnerbros.co.uk/games and https://community.wbgames.com/
2.2.3 Reference 2: Applying Goal-Oriented Action Planning to Games by Jeff Orkin

Mr Orkin wrote many other articles about planning in games, such as this paper Orkin and Productions [2013]. It mainly discusses about GOAP planners in games with the example of the one implemented in the specific No One Lives Forever 2 game project of Monolith Productions. This paper explain what a GOAP planner is, how it can be implemented and especially how it was used in NOFL2. It defines terms and benefits for the development of this technology, but also bring awareness about the implementation considerations needed for such projects. For instance, the author explain the importance of the world representation.

This article complete and overlap with the Reference 1 previously stated, from another game implementation. It is interesting to spot the possible differences but moreover the inescapable features to consider for GOAP planners.

2.2.4 Reference 3: A Real-Time PDDL-Based Planning Component for Video Games by Eric Jacopin

This paper co-written by Eric Jacopin analyses the usage of PDDL in a real-time game environment, Bartheye and Jacopin [2009]. This work greatly discuss about the use of PDDL in video games, and the place this language can occupy in future games. It argue about the importance of polishing the development of PDDL before any further usage in the game industry to facilitate the introduction of automated planners as a "must have" technology, being aware that this language could become inevitable shortly. The authors conclude that "it is important to ease each step in the mastering of PDDL, not by providing new AI Planning features, but by first reinforcing the software engineering features of PDDL and second by introducing plan execution features into PDDL".

2.2.5 Talks with Mr Jacopin

At the beginning of the project, Mr Jacopin and I were put in contact by Mr Orkin. We talked together about the implementation possibilities.

Mr Jacopin and I had discussions and talks about the way planners are implemented in games. He advised me about the different steps I could use to create my own planner. I decided to follow his steps. I first should to create the different actions available for

\[https://www.lith.com/games/no-one-lives-forever-2\]

\[https://fr.linkedin.com/in/eric-jacopin-6796111b/en\]
my agent. Beside this process, I should create a planner and test the algorithm outside the game environment. Finally, I would need to combine the game environment and its world state with the planner to test it in "real conditions" and adjust it. Mr Jacopin confirmed that GOAP (goal oriented planner) are the most used and efficient planners for such projects.

2.3 Unity Engine

As developing a prototype of a game environment in order to test the planner, I decided to make the use of Unity Engine. Unity is a game development platform for 2D and 3D games. The following sections will briefly present the important features used in this project.

2.3.1 Unity terminologies

The main feature in Unity is the use of GameObject to define elements on the screen. The can represent characters, HUD or obstacles. Scripts can then be linked to GameObjects in order to change their behaviors or interactions. Every Unity script derives from a MonoBehaviour class, the base class, that should be explicitly apply when using C#.

I made the use of Colliders to handle collisions with obstacles and objects. In addition, I created a Navigation Mesh and Navigation Agents to differentiate characters and obstacles, and give the agents space to move in the world. Rigidbody and Mask Layers help the management of collisions and detection, such as "line of sight" effects. The detections are done thanks to Raycast that determine where is the closest object and what it is (agent, wall, object).

One of the key feature in my project is the use of Coroutines. Coroutines are IEnumerator functions that can run asynchronously with Update functions. They are perfect for asynchronous needs, such as waiting for an action to finish, checking for line of sight or re-run the planner while the game is running.

For more features, I referred to the official website documentation of Unity.
2.3.2 Unity tools

Unity disposes of a great assets store\textsuperscript{18}. Assets are pre-made and reusable codes or GameObjects, free or paying, that could be added to Unity projects from the same version. They help developers in tasks they would not spend too much time on, such as character design or sound effects.

2.4 In Closing

PDDL and AI specific languages, although well developed and sometimes already used, are not recommended for real-time implementations in video games environments. Their complexity are time consuming for developers and similar performances can be achieve with easier methods, such as search algorithms in goal oriented planners. Automated planners are alike path-finding search as they search for states, to produce given plans or actions, in space. This project was conducted with this set of methods as guide-lines to fulfill objectives and goals. To this end, I chose to make the use of Unity Engine as development environment, using C\# as language for my own coding of the planner. The GOAP principles were used to implement such planner.

The next chapters will discuss about the fulfillment of this project and its achievements.

Report written by —
Margaux Lussiana

\textsuperscript{18}https://assetstore.unity.com/
Chapter 3

Ethics and Risks

3.1 Ethical Approval

This project is conducted by a single student. Most of the testing and evaluation steps will be done by the student herself. If external participants are engaged, the following information about them will not be required: any racial or sexual mentions, psychological questions, personal information. The main part of the testing process is internal and conducted by the student herself. A small number of people may have to participate for a second evaluation where they will be asked to perform a simple in-game test and give subjective feedbacks. Objective feedbacks could also be provided. No personal investigation would be done on those people. In addition, the project does not have the purpose for confronting people with a frightening or disruptive environment.

The ethical form for this project has been submitted and approved by the supervisor.

3.2 Risks Assessment

The evaluation of this project does not require any special equipment other than a laptop and a mouse. The testing environment is similar to any in-door office environment and does not present any kind of unusual risks for the person testing the project. Also, if external people need to evaluate or test it, all equipment will be provided by the student, along with assuring an in-door office-like environment such as a house or a classroom.

The risks are stated and validated in the Risks Assessment referred in the Appendix C, figure C.1, by the student in charge of the project and her supervisor.
3.3 In Closing

This project does not contain any risks for the people developing and testing it, nor any ethical issues concerning them. Therefore, the project and its evaluation part can be conducted without concerns.

Report written by —
Margaux Lussiana
Chapter 4

Development of the Project

4.1 Steps

This project required several steps in order to be carrying out well. The different requirements to fulfill this project are:

- **Prototype a game environment:** suitable for the purpose of the project
- **Build a planner:** choosing and implementing the algorithm
- **Build an AI:** organizing the planner around the different tasks available to the Agent
- **Play-test the game:** play testing the game environment to spot any bugs that can interfere with the testing of the Agent’s behavior
- **Evaluate the planner:** using different algorithm properties to get the best results
- **Test the AI:** play testing the whole game to evaluate the Agent AI

These requirements can be divided into two main sections, the development part and the evaluation part. The first one focuses on the design and coding of the system while the second one evaluates the system in different ways. The final objective is to properly fulfill the main goal of the project and answer the hypothesis questions.

4.1.1 Development

The development part of the project is split between the game environment and the AI development steps. The first task is to create a prototype for a game environment in
order to develop and test the future AI. The rules are adjusted all along the development for a fair balance. Many Unity properties are used to ease the conception of out of scoop elements, such as the map, the motions and physics. The agents, both playable and unplayable characters, are designed at the same time, but the NPC is further developed to be the host for the future AI and planner. First are created the agent’s tasks, such as firing, then the planner is articulated around the existing tasks and modeled so that the agent can autonomously make choices and do actions depending on the world state. The final step is of course to incorporate the planner and AI inside the FPS game environment in order to test it in play conditions.

4.1.2 Evaluation

The Evaluation is split into two main parts: a quantitative test and a qualitative test. The first one aims at showing the performances of the planner, and more specifically its suitability for real-time video games. It would be performed on several runs with different parameters. The second one aims at measuring the level of involvement of the player with this particular form of NPC.

The quantitative test stands for the internal design and implementation of the AI, while the qualitative test allows a more free appreciation about the concept of AI in games.

4.2 Structure

4.2.1 Planner and Language

GOAP is a complete and sufficient planning system. That is the main reason why I chose GAOP to conduct the project. It is often used in games that require quick and simple planning, such as real-time planning. It is also pretty straight forward to implement and intuitive for developers used to traditional object-oriented languages used to program games, such as C#. The choose of a GOAP system would allow me to clearly show what planners in games are, along with the real-time aspect and the implementation of search algorithms in games environments.

For this project, I chose the object-oriented approach as it is more natural when handling video games. However, I made use of the knowledge of PDDL I have gathered so far thanks to my previous work by using the same knowledge representation logic and adapting it into an object-oriented language (C#) for the implementation in games codes.
4.2.2 Applications and Tools

This project is made on Unity Engine in order to provide a simple game environment for testing purposes using C# language. Unity allows a great management around the creation of video games and can be used with an object-oriented language (C#). For the purpose of this project I aim at creating a planner and its settings in order to specifically design it for the given case. All of the code based around the AI and planner are my own.

C# scripts are coded using the IDE Visual Studio\(^1\) on Windows. It is directly linked to the Unity project and simplify the debugging process.

Unity presents a large Assets store to help carrying out projects. This project makes the use of three different free assets: SciFiHandGun\(^2\) is used for the gun visual, Original Bricks Textures\(^3\) for the walls and floor visuals, and Weapons of Choice\(^4\) for the weapon’s related sound effects.

Note that the sound effect displayed when picking up a health pack is a free-to-use clip from the audio library of YouTube\(^5\), called Wood Plank Flicks from the cartoon category.

4.2.3 Scripts

This project is structured with eight scripts attached to the Unity project. Four of them are used for sound effects: OutAmmoEffects produces the sound when the character is running out of ammo, reloadingEffects displays a sound when reloading, FiringEffects displays a sound when firing, HealthEffects produces the sound when an agent regain health points. Two scripts are specifically involved in the playable character code: CamMouseLook handles the look of the player through the manipulation of the camera, FPSController refers to the commands and physical aspects of the playable character. The last two scripts are related to the AA (bot) and its planner: EnemyScript is the bot script managing its physics and actions, AgentScript hold the planner system.

The code file submitted with this report includes three different Unity projects: GameLevel - BFS, GameLevel - AS and GameLevel - debug. The first two refer to the designs of the planner and contain those specific codes (eg. Best-First search algorithm in GameLevel - BFS). The GameLevel - debug project contains the debugging view used during the

\(^{1}\)https://visualstudio.microsoft.com
\(^{2}\)https://assetstore.unity.com/packages/3d/props/guns/sci-fi-futuristic-hand-gun-90249
\(^{3}\)https://assetstore.unity.com/packages/2d/textures-materials/brick/15-original-bricks-textures-72427
\(^{4}\)https://assetstore.unity.com/packages/audio/sound-fx/weapons/weapons-of-choice-free-101807
\(^{5}\)https://www.youtube.com/audiolibrary/soundeffects
project in order to test the actions via buttons or to test the plans on an immobile target, with the BFS algorithm.

4.3 Game environment

In order to properly observe and test the behavior of the Agent, I prototyped an FPS-like game environment. The player would face the Agent in an arena designed for this project shown in figure 4.1. The two agents freely walk around the map with obstacles and objects and have to kill each other to win. The view of the player is a FPS view.

I also created another view setting allowing me to test and observe the intelligent Agent during development, but the arena and objects remain the same.

The player is placed in an arena with an agent, four ammo packages and one health pack that re-spawn. Both the agent and the player are equipped with a loaded gun and need to kill the opponent. The re-spawn positions are random on the map. The scores can be recorded to compare the player and the bot performances. However, the aim of the project is to discuss about the use of such AI technology in real-time games, that said the player has no obligation to kill the bot even though the bot will always try to kill him.

4.3.1 Game properties and mechanics

This game uses several Unity properties, functions and mechanics very helpful to develop agent’s behaviors with a certain ease and speed. Unity allows the user to attached many mechanics and technologies to GameObjects, and used them easily in the scripts. A
**GameObject** can be anything on the scene, but for this project it is mostly visible objects such as characters, walls or HUD.

The **Collider** mechanic is entirely managed by Unity and allows collisions detection between objects. It is also used with the **Raycast** mechanic which permits the agent and the player to aim and fire at a target, and validate or not the shot. **Raycast**’s properties give the first obstacle hit by the shot (ray) in a specific direction thanks to the **Colliders** attached to the **GameObjects** on the scene.

Figure 4.2 shows the process to detect a shot using **Raycast**, on the bot side. The function updates the current state by updating the corresponding variables - further explained in the section dedicated to the planner and the AI below. It also sends the information to display a sound to the specific scripts if the conditions are met.

![Fire function from the agent side](image)

*Figure 4.2: Fire function from the agent side*

**Colliders** are attached to any physical objects on the scene, such as the characters, the walls or the packages. Another mechanic is necessary to easily manage the collision between the floor and the characters on the map without using collision checking, which can be expensive: the **Rigidbody**. It needs a **Collider** attached to the **GameObjects** to work correctly, and it basically handles the physics properties when it comes to object motion and position with gravity. When the objects collide with the floor, the **Rigidbody** manages the physical reactions.

Another mechanic that Unity offers to manage the motion and the space is the **NavMesh**. This project makes the use of **NavMesh**, **NavMeshAgent** and **NavMeshObstacle**. The first one allows the creation of a walkable space in the scene for navigation mesh agents, the second one represents the agent that can navigate on the navigation mesh and the last one allows the creation of obstacles, cut-out or carved in the mesh. For instance, the walls are cut-out obstacles as they never moved and are meant to stop the progression of the agent, but the packages are carved obstacles as they are interactive objects. Both the agent and the playable character are **NavMeshAgent** in order to make them able to use the mesh on the floor and interact with obstacles. The **NavMesh** properties also
allow simple path-finding actions for the agent to find a way around using the mesh, as
path-finding through the space is not an objective of the project. Those features are
very useful and ease the FPS-like game and AI developments.

Figure 4.3 shows the code lines for the collision between a character and an object,
such as ammo package or health-pack. The collision is detected on trigger by the On-
TriggerEnter function that manages the detection on its own. There is no need to call
the function in the Update function as it acts as an event checker. Once a collision is
triggered, it first checks for the tag of the collided object in order to filter the collisions
detected. The second condition in each if statement allows the management of sound
effects, values updating and availability of the object. The Coroutine ”waitTime” starts
a timer of X given seconds in order to permit the re-spawn of the object.

```csharp
//trigger collision:
void OnCollisionEnter(Collider other)
{
    if (other.tag == "ammo" && Ammo < 4)
    {
        Package = other.gameObject;
        Ammo = 4;
        GameObject.Find("Reload").GetComponent<ReloadingEffects>().flagAmmo = true;
        other.gameObject.SetActive=false);
        StartCoroutine("waitTime", seconds);
    }

    if (other.tag == "health" && Health < 10)
    {
        Package = other.gameObject;
        Health = 10;
        other.gameObject.SetActive=false);
        StartCoroutine("waitTime", seconds);
    }
}
```

Figure 4.3: Detection of collisions with OnTriggerEnter method

4.3.2 Visual features

This project provides three different HUD (figure 4.4). One is attached to the player
point of view, showing its health-bar and ammo-bar. It needs the first camera to be
display. A second one was used during development time for testing purposes. It displays
click-able buttons to trigger the agent’s tasks. The use of the second camera is necessary
in order to have an overview of the board and the agents. In that case, the bot and
the target are used. A third HUD is used to display the bot information such as its
health-bar and its current state. It is shown with both cameras.

Figure 4.4 shows the three different HUD used in this project, in previously stated order.
Those HUD were done with canvas features in Unity. This feature also allows to display sound effects. The fire, reload and out-of-ammo sounds are attached to canvas in order to be used by the scripts. The bot and the player have their own canvas.

4.3.3 Game Characters

The playable game contains a playable character in a first person view and an artificial intelligent agent to play against. The playable character is directed by the player. The player can move in any directions using the keys Z, Q, S, D, (keys peculiar to the specific computer on which the game was developed) look up and down at 180 degrees, and look left or right at 360 degrees.
The player is equipped with a gun and four ammos. The middle of the screen is marked with a green point, representing where the gun is shooting (see figure 4.4. The player needs to fire at the Agent two times in order to win the round. The Bot needs to shoot the player two times in order to win the round. The player can crouch in order to move slowly. The bot can spot the player presence at a certain radius to mimic the detection of sounds produced by the player.

In the testing set of this prototype game using a MOBA-like camera, the bot faces a "target". This target is used to trigger the agent abilities and test its responses during development, such as the behavior "go to target". The target can be displaced (by pressing P to spawn it randomly) and can be killed by the agent. When killed, it re-spawns randomly on the navigation mesh. It also has a Rigidbody and a Collider.

### 4.3.3.1 Creation of the Artificial Agent’s tasks

This part of the development occurs with the second game set, with the MOBA-like camera over the map, the agent and the target. This part was very important to implement the behaviors of the agent, that I will call Actions, that will be used in the planner. It permitted several tests and improvements during the development, especially about interactions between the Actions (such as priorities).

The target created for this settings is helpful for me to determine how good the actions are performed and test them in different made-up situations. Also, some buttons to press control the different actions available for the agent, such as firing or going to the target. Pressing any buttons would lead to the immediate response from the agent and fulfillment of the action. Any action can take over another in execution, as it is only for testing purposes and not yet planning. The tests only involve the Actions and not the reflexion of priorities.

The agent itself is exactly the same as the one used in the FPS game, in terms of GameObject and script. For this specific part of the development, the script was adapted to suit the "target" aspect, meaning it is not a playable game with a second character but only a test set for the agent’s actions and their interactions. However, the final game would contains the implementation of the entire planner in the game for further tests.

There are five actions available with buttons as shown on figure 4.5:

- **Fire**: immediate fire along the forward direction (Vector3) of the agent.
- **Go to target**: use the GoTo Unity function to directly go to the target position, anywhere on the map. The destination is updated when the target is moved.
• Find ammo: go to the nearest ammo spot. They are fixed and known on the map.

• Find Health pack: go to the health pack spot. It is fixed and known on the map.

• Patrol: Perform a patrol between four specific points. Go on until another moving action is instructed.

![Figure 4.5: Test scene for the agent actions during development](image)

The ”fire” action is a hit-scan capacity, meaning the shot is a direct and immediate hit at anything in front of the bot. If the target is hit and close enough, it receives five damage points. If the target is out of range, it doesn’t receive any damage. The fire action doesn’t interrupt any of the other action, it can be perform simultaneously with a moving action. It uses the same Raycast principle as the fire capacity of the playable character.

The ”go to target” action knows the exact position of the target anywhere on the map at anytime. If the target moves, the known position is also updated and the agent will follow the new one. It interrupts any other moving actions when the button is clicked.

The ”find ammo” action and the ”find health pack” action interrupt any other moving actions and order the agent to go to the nearest object using Euclidean distance between the agent position and the destination position.

The ”patrol” action is developed in a Coroutine function (see figure 4.6) that allows it to run continuously until it is ordered to stop (by another moving actions). It interrupts any other moving actions. When the Patrol button is clicked, a flag is trigger to start the first Coroutine: ”EnterPatrol”. This Coroutine parses a list of patrol points set on the map. Each time it selects an element, the second Coroutine starts. ”WaitForPatrol” allows the agent to move to the point until it is reached. Once reached, the Coroutine stops and the ”EnterPatrol” function goes ahead by selecting the next element. This system goes on until another moving action is performed.
4.4 Creation of the Planner

The planner part is made in another script, called AgentScript, in order to better dissociate the enemy bot and its intelligence. That way, it helped in the testing process, both in FPS version and overview version, using the same planner code.

The planner uses a class, Node, in order to represent the different actions possible. Each Node contains a state variable which relate the state of world for each Node. The algorithms use the nodes to find a plan through all the different possibilities.

4.4.1 Knowledge Representation

The knowledge representation is one of the most important part when creating a planner. A good representation of the world can greatly simplify the implementation.

I created a Node class containing the different characteristics of an action, such as preconditions and effects. Each action is created as being a Node and defined in the Start function of the script. Each Node also has a State dictionary of string and boolean pair values to represent the different variables of the world and their states at a given point in time. The State dictionary, apart for the start Node, is attributed when searching for successors and so needed to know what effects an action can apply to the world. Each Node also keep tracking of its own parent Node when being created. The parent variable is used by the planner to return the plan at the end of the process.
The class method Successors is called by the planner to find the successors of the current node in order to create a path through the world space leading to the goal. It compares the preconditions of each node to the current world and, if they are satisfied, returns the node as being a successor. The class method IsGoal checks if the current node satisfies the goal conditions. The class methods Cost calculate the cost of each action, also called heuristic, depending on different parameters I arbitrary chose for testing purposes.

I also created a CustoMethod class. It allows me to create custom methods for any kind of purpose, such as a Pop method applicable on Lists, directly inspired from the stack method, shown in figure 4.7.

![Figure 4.7: Pop method applicable for Lists, custom method.](image)

The world space is defined with six boolean variables representing different states of the bot: Ammo, Health, SearchingTarget, TargetInSight, GoToTarget, TargetHit. Both AgentScript and Enemyscript handle those variables in order to keep tracking of the current world state when re-planning. Hence, the goal is expressed as one of those variable in order to change its value. For instance, when the bot is running out of ammo, the planner’s goal is "Ammo == True" to signify that the bot needs to find some ammo.

### 4.4.2 A* Algorithm

The planner uses an A* algorithm to create a plan through the world space. A* is a very well known and used algorithm, especially for path-finding. However, it fits perfectly the purpose of this project, as planning is a path-finding through world states.

A* works with an OpenList and a CloseList of objects. The OpenList contains the possible Nodes, the CloseList the pruned ones. Each time the planner test a current state, it searches for possible successors that could lead to the goal and prunes the other branches of the search. The A* algorithm of this project is a forward algorithm, meaning
it starts from the current state and find a way to the goal (bottom-up). Each time the algorithm finds a possible successor, it attaches its parent. At the end of the process, when the planner finds the goal conditions to be fulfill, it stores the current Node (Goal) in a List and add successively the parents as shown in figure 4.8. It continues until reaching the very first node of the plan, having no parent as it is the starting point of the search. Finally, the List containing the plan is stored in a global variable, ready to be used in another script or by another function.

```csharp
if (CurrentNode.IsGoal(goal))
{
    while (CurrentNode.parent != null)
    {
        Solution.Add(CurrentNode);
        CurrentNode = CurrentNode.parent;
        yield return new WaitForEndOfFrame();
    }
    continue;
}
```

Figure 4.8: Creation of the list containing the final plan.

As the A* requires some time to process, the use of a coroutine is mandatory. It allows the entire project to continue running while searching for a plan. The entire code for the forward A* algorithm can be found in Appendix D, figure D.1.

### 4.4.3 Best-First Algorithm

I decided to implement a simple search algorithm for testing purposes. It is a Best-First search (BFS) algorithm which uses a single cost value instead of heuristic, gScore and fScore. BFS algorithm also searches a path through nodes in order to satisfy a goal, using the knowledge representation of the world states. It uses the same function as the A* search to find possible successors when planning. The entire algorithm is shown in Appendix D figure D.2.

Best-First search is a simpler algorithm, often said to be less effective in most cases than the traditional and commonly used A* search. However, I think that having two different designs for the planning system can help in discussions and conclusions when evaluating the planner. Moreover, it can help outlining specificities of the project.
4.4.4 Implementation of the AI

Implementing the planner inside the game requires to link together the game environment and the planner variables. That is done with a dictionary of string and boolean pairs of variables, related to the six parameters of the bot and their respective values. Each of those values are modified in real-time by the EnemyScript whenever an event happens to change them. Those changes are tracked to verify the application of the plan and the evolution of the world. Whenever the AA needs a new plan, the EnemyScript sends the dictionary representing the current state to the AgentScript, along with a dictionary of one string and boolean pair representing the new goal of the planner.

In order to send those dictionaries, I need to transform them as a list of objects and interpret them once received. The first object of the list is the states dictionary and the second one is the goal, as shown on figure 4.9. The left method is the sender in EnemyScript, the right one the receiver in AgentScript.

![Figure 4.9: Communication between two script in order to keep tracking of the current state and re-plan.]

4.4.5 A temporary testing and debugging environment

The implementation of the AI in the game environment needs a specific testing system to ease the integration. The camera used is presented in figure 4.4, middle view. The planner is run and the bot executes the sequence of actions on its own. However, the playable character is disabled and replaced by the single target character. Note that the action "HP" was never triggered, as the bot was never hit by the target, but the "Ammo" function works similarly to the "HP" one.

This environment helps spotting bugs and understanding the function mode of the bot - and the planner - in real conditions. It allows me to adjust the planner code and the link between the two scripts to fits better the game requirements.
4.4.6 Adaptation with the actual game

The testing environment is handier than the actual game, especially because the real-time aspect is less important as the target could not properly move on its own and initiate behaviors. On the contrary, the playable character brings challenge to the bot and the planner. The bot would not only have to react to the player’s actions but also to react to its own variables changes, such as the lost of health points. Therefore, I implemented a tracking system for the environment and bot variables that substitutes the flags used in the debugging sessions. In addition to the list of states, two coroutines keep tracking of the ammo and health variables. The figure 4.10 shows the code used in those coroutines and the re-planning system that gives the correct reason for asking to re-plan. The one presented only triggers the ammo variable, but the second one works similarly and independently triggers the health variable. The GoalReason variable is a dictionary of only one string and boolean pair of variables that represent the goal state and its value of the next plan to achieve. Once given to the AgentScript, it becomes the new goal for re-planning.

```csharp
//COROUTINE: Check for the agent ammo and trigger a "replan" if necessary
private IEnumerator CheckValuesAmmo()
{
    while (true)
    {
        if (enemyAmmo == 0)
        {
            Ammo = false;
            states["Ammo"] = false;
            GoalReason.Clear();
            GoalReason.Add("Ammo", true);
            AskReplan(GoalReason);
            yield break;
        }
        yield return new WaitForEndOfFrame();
    }
}
```

Figure 4.10: Keep tracking of the Ammo value in order to trigger a re-planning if necessary.

4.5 In Closing

The developing part is well divided between the creation of the prototype of the game and the implementation of the planner. The most challenging part to me was the integration of the planner within the game environment with all its constraints. I had to try several ways of communicated between the two scripts and several type of variables that could
be easy to use, both with the scripts but also with my knowledge representation and the
design of my planners.

Report written by —
Margaux Lussiana
Chapter 5

Test and Evaluation

The aims of this project are to measure and understand the real-time aspect of planners in games as well as the player involvement when facing artificial agents.

The evaluation part is divided into 2 sections to conduct the validation of the two following hypotheses:

- **Hypothesis 1:** GOAP planner is an adequate technology for real-time applications, such as video games.

- **Hypothesis 2:** Interaction with artificial agents are mostly enjoyed by players and have an impact on their involvement in the game.

The quantitative testing requires test runs and values recording, comparisons and analyses to validate, or invalidate, hypothesis 1. This evaluation part is entirely done by the student in charge of the project.

The qualitative testing requires voluntary participants to test the project by playing against the agent and answer few questions. Only relevant information about participants’ thoughts and feelings are recorded in order to validate, or invalidate, hypothesis 2. This evaluation part is performed by external participants and analyzed by the student in charge of the project.

5.1 Quantitative testing

The quantitative testing part is used to gather several values and variables, and try to analyze them. The recorded information are used to investigate on performances of
GOAP planners, the specific A Star search algorithm used, the specific Best-First search algorithm used, and the different values inside the planners to manage the planning system.

5.1.1 Conditions of testing

The quantitative test includes the comparison of two different heuristics for the A Star algorithm, the comparison of two different search algorithms and an analysis of real-time performances. They are conducting on ten runs of the game of approximately 30 seconds each. The following relevant information are recorded:

- **Frames per second:** to outline the real-time aspect
- **Planning execution time:** to outline the real-time aspect
- **Length of the plan and actions:** to outline the possible efficiency/quality of the plan
- **Reason for re-planning:** to outline the integration of the planner within the video game environment

Whenever a planner is unable to re-plan, the fail is recorded but it continues to run to operate the next re-planning demand. If the re-planning is totally impossible, the run is stopped and started again in order to reset the planner.

5.1.2 Planners

The first planner tested is an A* search with a heuristic function, distance between two nodes and cost since the beginning of the plan. Two different arbitrary chosen heuristics are tested for performance comparison purposes. The distance between two nodes doesn’t really apply here as successor nodes are always the next possible nodes, accessible from the current node. This function has a constant value of 1 to represent this design. The cost since the beginning of the plan is represented by the addition of the heuristic of next node and the cost from start of the current node, which is a mix between a known value and an estimated value. I just want to mention that those variables and functions can have an impact on the planner, but I will not describe them further as it is not the main objective of the project. The complete A* algorithm is presented in Appendix D, figure D.1.
The second planner uses a Best-First search algorithm with a single cost value. This implementation is specific to this project and is used to compare real-time performances depending on the design of the search algorithm. A subjective comparison of plan quality can also be done between the two algorithms. The complete Best-First algorithm is presented in Appendix D, figure D.2.

5.1.3 Heuristics

The first heuristic used in A* algorithm is always equal to 1, meaning it doesn’t prioritize any actions. It is mainly used as a reference, both for comparing heuristics and focus on other aspects of the planner. The second heuristic is represented by the number of preconditions needed to achieve an action.

For the purpose and aim of this project, I chose simple heuristics to compare results. I am aware that better heuristics - that said, that fit better the knowledge representation or the planner design - can lead to better performance results. However, the goal of this project is not to find the best design but to show evidence of a possible use of planners in real-time video games. I think that simple and commonly used heuristics help to understand and analyze the fundamentals of this technology that make real-time applications possible, or impossible.

The Best-First search algorithm uses a cost value always equal to 1, in order to compare it with the first A* algorithm (heuristic equal to 1).

5.1.4 Comparisons and statistics

Different comparisons are done with the several recorded data. T-Tests are performed between different sets of data from the planning execution time values to answer the null hypothesis and ensure the significance of their differences.

The planning execution time datasets are compared together, for the three different version of the planner and using the A* cost#1 as reference, in order to outline the real-time aspect of them. Frames per second, minimum and maximum values per run, are compared to emphasize another aspect of real-time games: playability. Mean length of plans, and possibly actions per plan, could be compared to support comments on efficiency of the planners.
5.2 Qualitative testing

Video games represent an adequate mean of testing AI, planners and people reactions to AI in a simple environment, easily understandable and familiar for some people. This qualitative testing is used to gather appreciations from people about the effectiveness of the bot and its impact on players. However, participants can have different previous experiences with games and/or AI. Those information are recorded in order to interpret their comments.

At least 5 different participants took part of the evaluation. The recorded answers of the participants are used to investigate on the participants’ feelings and thoughts when facing the intelligent agent. All of these information are subjective comments and appreciations that need to be discussed within a specified context, and always compared to the knowledge of the gaming environment and AI technologies of the participants.

5.2.1 Progress

The participants have access to the questionnaire before the testing part so that they can prepare to answer the questions. This process can add some bias to the answers. However, for the purpose of this project and this evaluation part, I think that a comprehension of what is actually tested is beneficial over the bias. Indeed, this part aims at measuring the possible efficiency of the AI by gathering subjective comments about the agent intelligence and the involvement of the player when interacting with it. Therefore, participants will not be specifically asked to ”win the game” by killing the bot but rather to investigate on one or both of the aspects previously stated. Reading the questions before the evaluation is a mean for them to understand their goals in this experiment.

Participants are asked to play at least 1 minute of the game, until they want to stop playing. They play with two different versions of the code. However, the questionnaire is oriented around the Best-First Search version; the A* version (using the heuristic equal to 1) is used to subjectively compare the quality of the plans, depending on the planner design. This comparative question aims at showing the differences between technologies used in order to emphasis the possible impact of several factors on the quality of plans.

5.2.2 Questionnaire

The questionnaire given to the participants is shown in Appendix E. It gives them basic rules and information about the game and what their are asked to do. Some questions are closed-questions with simple answers such as ”yes” or ”no” to clearly state some
features. Most of the questions are opened-questions with large blank spaces to fill by the participants to explain in details their thoughts and appreciations. Opened-questions permit theme analysis to outline common comments written by several participants. The scores and play-time of participants are also recorded and used in analysis if relevant enough.

5.2.3 Theme Analysis

A theme analysis is conducted on all the participants answers to outline any relevant information about the agent’s behaviors. I analyze them and use a critical sens to discuss about their importance, subjectivity degree and impact for this project development and conclusion. The theme analysis can show and gather evident and common thoughts, sometimes predicted before the evaluation, but also unpredictable answers and comments on unasked features.

5.3 In Closing

This chapter stated the development of the evaluations, the values recorded and the possible comparisons. The next chapter shows the results of those evaluations and conclude about the hypotheses of the project.

Report written by —
Margaux Lussiana
Chapter 6

Results

The goals and important information about the evaluations of this project are stated in Chapter 5. The decisions, choices and conditions of those evaluations are always considered when analyzing the following results.

6.1 Quantitative Results

The A* search runs gather 89 (cost #1) and 90 (cost #2) executions of the planner, for respectively 4:15 min and 4:03 min of game-plays. The BFS runs gather 67 (cost #1) executions of the planner for 5:07 min of game-play.

6.1.1 T-Tests

The T-Tests parameters are a significance level of 0.05 (5%) and a two-tailed hypothesis. They try to invalidate the null hypothesis, meaning try to find a significant difference between the 2 sets of data. If the P-Value of the T-Test is inferior to the significance level (0.05), the 2 sets are considered different enough and their comparison is acceptable.

The first T-Test is performed between the A* search, cost #1, and the A* search, cost #2, planning execution time values. The P-Value is equal to 0.14942, which is greater than 0.05, meaning that their is no significant difference between the mean distribution of these two datasets. Therefore, I can not compare them to prove any performance improvement.

The second T-Test is performed between the A* search, cost #1, and the BFS, cost #1, planning execution time values. The P-Value is equal to 0.010682 which is less than 0.05. Therefore, the comparison of the two datasets can prove a performance improvement.
between these different designs of planner. The following graphics show the planning execution time for all the different plans produced during the tests for each planner. The red line shows the mean value. We clearly see that the mean value of BFS, figure 6.1, is smaller than A* mean value, figure 6.2. There is a difference of about 20ms.

**Figure 6.1:** Planning Execution Time Values, BFS cost 1.

**Figure 6.2:** Planning Execution Time Values, A* cost 1.

*NB: The sets can have different length when performing a T-Test.*

A value of 0 indicates that the planner was unable to find a plan. I can notice that most of the time, whenever this case happens, the planner succeeds to plan when trying a
second time. As a subjective analysis of the game-play, most of the fails while planning were imperceptible. Over the three different planners, five fails conducted to the end of the current session as the planner was totally unable to re-plan for several attempts. The majority of those fails happened with the A* search cost #2 test sessions.

6.1.2 Real-Time analysis

As shown in the previous section, the BFS planner is the more efficient for real-time planning execution in this specific project. It is important to look at different variables that can confirm the real-time usability of the planner.

The execution time of the planner - of the search algorithm - needs to be short enough to give the illusion of quick human-like reactions from the artificial agent. The average reaction time of the human brain seems to be around 250ms for visual stimulus and 170ms for audio stimulus, as stated by this vulgarization post about neuroscience\(^1\). As video games are all about visual and sound effects, this gives a great comparison for the execution time values recorded before. However, muscle reactions can differ from neural reactions. This website\(^2\) gathers more than 50 millions click reactions records and provides an average of 273ms of reaction, specifically on electronic devices. Together, these 3 average values of reactions time for a human brain can help understanding the average execution time of the planners. The means for BFS and AS with cost #1 of execution time are both noted on figures 6.1 and 6.2; the mean for AS with cost #2 is equal to 100.98 ms. We can clearly see that they are all less than the average human brain reaction time, which means the planners can compete against humans in real-time. Moreover, none of the recorded values exceed 273ms for the three versions.

In real-time video games, especially on-line games, frames per second are very important for the player to correctly see events happening and react in time. It is generally admitted that 30 frames per second are the minimum admissible for real-time video games. A comfortable frames per second value is 60. A stabilized number of frames is also very important for the fluidity of the game. The figure 6.3 shows the different records of frames per second per run. The minimum values never drop under 60 frames per second, which is commonly said to be a standard frames per second value in real-time games and on-line games. Average drop of frames per seconds in less than 10, which is considered stable enough.

\(^1\)https://backyardbrains.com/experiments/reactiontime
\(^2\)https://www.humanbenchmark.com/tests/reactiontime/statistics
6.1.3 Subjective quality of plans and integration analyses

During the quantitative tests, I recorded the length of the plans as well as the actions to perform inside the plans. The Best-First Search planner gives a maximum plan length of three actions, while A* Searches give a maximum length of four actions. Knowing that my project offers five different actions to perform, the maximum length of the plans seems in all cases pretty acceptable for real-time planners. However, the plans’ quality seems to be different depending on the search algorithm used.

For instance, the majority of the plans of length three for BFS give the following actions to perform: "Patrol", "Go to target", "Fire". On the other hand, the majority of plans of length four for both A* searches give the following sequence: "Patrol", "Fire", "Go to target", "Fire". The first sequence might seem more intelligent and direct, as the bot will go to the target before firing, assuring a proper aim at the target. The second sequence gives a reaction that seems more based on reflexes, as the bot will fire first before trying to really aim at the target. This on seems less intelligent, at least to observe. However, it seems to me that the second sequence give a more human-like
process for its reactions, as human players often react quickly with reflexes, and then try to aim at them once they assimilated more information about their environment or enemy.

A shorter plan is useful for real-time applications that need to re-plan often depending on their environment, but I still believe that the quality of plans I observed during the tests are subjective to the observer’s point of view and experience. In that sense, both of the previous sequences of the plans are acceptable within the scope of this project for real-time reactions simulation.

On top of that, along with the length of plans, the reason for re-planning was recorded. In all and every cases, the reason given by the EnemyScript in charge of maintaining the world states updated and the new goal pursued by the planner in the AgentScript were exactly the same. This shows that my knowledge representation and the communication system between the two main scripts worked well in the context of the project. This validate the integration of the planner inside the prototype of the game.

## 6.2 Qualitative Results

The qualitative test gathers at least five questionnaires filled by participants. I will discuss their answers and subjectivity, and compared them with the possible causes to those answers. Also, my own subjectivity can be added in the discussion, especially when comparing their answers with some recorded variables from the quantitative test.

### 6.2.1 Theme analysis

First of all, the questionnaires outline two main types of game-play that participants performed during the test. This can help understand some of the values or comments later on. The majority of them tried to play by rules and kill the bot, which means "win the game". However, some of them only tried to trigger the bot and test its possible reactions. Some participants tried both of those game-play styles.

The theme analysis on the participants’ questionnaires outline three main ideas: the involvement of the player, the intelligence of the bot and improvement of the bot.

First of all, the majority of players tend to answer question II (E, E.1) with "somewhat disagree" for fun and "somewhat agree" for interesting. Their comments relate the interest they have to play with the bot and try to trigger it, which confirm their answer. Also, some of them discovered the AI technology through this test, which can sustain
their interest. Most of them spend more than 5 minutes playing the game which is 5 times more than the minimum required time. The interest with the technology could have lead them to play more, but some comments let me think that some participants had fun while playing: “the bot reactions are funny sometimes”, ”When the bot turned around I was surprised”, ”baiting the bot is funny, sometimes it looks stupid”. However, the highlights of the fun part seems to concern the bot and its reactions more than the game itself.

Words that come back often in comments concerning the behavior of the bot are the following: stupid, funny, predictable, not very smart. However, the same participants already played video games before, which means they could have other experiences with bot and AI in games. Those previous experiences could have raised their standards and made them more critical toward this project. Two participants never played video games before, or at least FPS games, and their comments were less harsh: ”the bot is smart enough to play”, ”I find that the bot reacted fast most of the time”. Though, the great majority of the participants said that the bot seemed too slow when performing the specific actions ”Find Health pack” and ”Find ammo”, which is probably caused by its constant speed movement and the fact that it doesn’t change its goal even if the player enter in line of sight of the bot.

The suggestion box for improvements of the bot was often filled up with the same comment about adding more possible actions. The more actions the bot can perform, the more smart it would look if, and only if, the design of the planner permits it. That said, for my specific design and project purposes, adding more actions was not relevant at that time and would have probably suggest a complete rework of the planner, knowledge representation or communication between scripts in order to keep the same real-time standards of the project. Some other comments suggested to change the priority system for the selection of the actions inside the planner, or to make the patrol action a bit more random to cover more space on the map. Most of the comments talked about the actions of the bot, and not about its direct reactions, or possible latency, which is a great point for the real-time aspect of the project.

### 6.2.2 Scores

The participants allowed me to use the recorded scores they performed against the AI. Everything apart, the figure 6.4 shows the different scores and play time of the sessions. It outlines that the player often wins against the bot, which means that the bot may not be enough challenging or balanced for average players. However, some of the players didn’t really try to win the game but played around the bot. As some of them describe:
"tried to follow the bot, gain its attention, bait it" or "test limits, test multiple scenarios such as avoid, ambush...". Therefore, those results may not reveal the actual challenge of the bot.

<table>
<thead>
<tr>
<th>Participant</th>
<th>AI</th>
<th>Player</th>
<th>Play time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>6:43 min</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>6</td>
<td>5:02 min</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>1</td>
<td>4:47 min</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
<td>6:12 min</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>8</td>
<td>6:36 min</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>4:13 min</td>
</tr>
</tbody>
</table>

Average    2.67        4.5   05:56 min

Figure 6.4: Table showing scores recorded during the qualitative evaluation among 5 participants.

### 6.2.3 Other comments

Some comments are more related to the game itself and less about the bot. The game environment presented few problems and bugs that didn’t impact the testing sessions whenever the participants were informed about them. Some of the players deplore the fact they couldn’t do some actions due to those bugs, even if it didn’t change their opinions on the project.

One of the participant had trouble finding the bot on the map and suggested a smaller map. This could be caused by the random re-spawn positions on the map after a death. However, this didn’t impact the overall testing process of the possible behaviors of the bot.

### 6.3 Achievements

All of the following achievements from the development and testing parts are completed at the end of the project:

- **Prototype a game environment**: suitable for the purpose of the project
- **Build a planner**: choosing and implementing the algorithms
- **Build an AI**: organizing the planner around the different tasks available to the Agent
• **Play-test the game:** play testing the game environment to spot any bugs that can interfere with the testing of the Agent’s behavior

• **Evaluate the planner:** using different algorithm properties, and different algorithms, to get the best results

• **Test the AI:** play testing the whole game to evaluate the Agent AI, with participants

As already stated in the quantitative results section, we can clearly see that the bot is responding correctly in real-time. Moreover, players didn’t comment anything on a possible latency of the bot.

The qualitative results section shows a clear involvement of the players in the game. The play-time is five times superior than the minimum asked, the comments show the participants engagement in the bot testing part, as well as improving it. Most of them found it at least interesting to test, some said it was fun.

Hence, the two hypotheses stated at the beginning of the evaluation chapter can be validated:

• **Hypothesis 1:** GOAP planner is an adequate technology for real-time applications, such as video games.

• **Hypothesis 2:** Interaction with artificial agents are mostly enjoyed by players and have an impact on their involvement in the game.

### 6.4 Conclusions

As already reported in the literature review, most of the video games using planners for AI use GAOP technology. Moreover, my own project using this technology validated the real-time goal. The majority of the participants to my evaluation were involved in the game and related this to their confrontation with the AI.

As the objectives of the project and the hypotheses of the evaluation parts are all validated, I can conclude that my project achieved its main goals, that were the real-time aspect of GOAP planners and the involvement of players against AI in video games. Hence, the hypothesis questions of the project can be answered:

1. **Is planning a suitable AI technology for real-time video games in terms of performance?** Yes, planning and especially GOAP planners are suitable technologies for real-time video games as demonstrated in this project.
2. How are intelligent agents impacting players involvement and fun in video games? Facing intelligent agents seems to engage players enough to satisfy the main goal of a video game: entertaining people.

Report written by — 
Margaux Lussiana
Chapter 7

Discussion

7.1 Project difficulties

This project had several problems and difficulties to overcome, that sometimes were not fixed.

When testing the game, I encountered a bug that made Unity crash when occurring: if the player try to hit the bot whilst it performs the Patrol action - meaning it hasn’t spotted the player yet - , the game freezes. I personally think that this is not caused by my planner code but rather by my use of coroutines. I discovered coroutines with this project and my knowledge about them is very limited. They are used for asynchronous functions, but doesn’t act like true parallel functions. Each time the update function of Unity is called, all the coroutines continue from the last breaking point reached ("yield return" instructions). However, I don’t know their running order, which had probably lead to multiple problems in my code that would need more time and further investigations to be fixed. I arranged the testing parts so that this bug doesn’t interfere with the goal of the tests anyway.

Another difficulty in this project, that was overcome, was the knowledge representation. I had trouble finding a good and easy to use KR when implemented the planner within the prototyped game. I had to try out several ways for expressing the world states. I think that there are as many KR as the human brain can imagine. In other words, the possibilities are limited to the human imagination. However, it doesn’t mean every possibilities are good or efficient for every project. The most difficult part is to find a correct KR for a specific project.
7.2 Future Work

This project can benefit from multiple adjustments and improvements:

- The heuristics and designs of the planners used could be deepened to reach some performances goals that were not defined in this project.
- The game environment can be more polished in order to give a more immersive experience to players and improve the impact of the AI on their involvement.
- The bot could offer more possible actions to develop its plans and give a better illusion of intelligence.
- The world states could have more variables in order to simulate more accurately what could happen in a game.
- Other technologies of AI or other tools could be tested for several comparisons, such as PDDL or machine learning.

Those features could be further developed after the submission of this master project for personal interests.

7.3 Self-Actualization

This project allowed me to go further in AI technologies I discovered during my first term in my AI courses. I am passionate by video games, and I really think that it is a great tool to test and develop AI technologies. This is the reason why I chose to implement a planner inside a prototype of a video game. I really wanted to investigate on planners and planning technologies, which I achieved during this project. I now wish to continue and test other AI technologies in future projects.

7.4 AI in games

Further discussions can be done about AI in games. I would like to emphasize the great usability of planners, and especially GOAP planners, in video games as demonstrated in this project. As already stated by Jeff Orkin, GOAP planners were used in F.E.A.R, Orkin [2005], and NOFL2, Orkin and Productions [2013]. Thanks to this project, I can clearly see the reasons why GOAP planners suit great in video games, moreover for real-time applications.
The use of a unified framework for AI in games is an idea developed in Safadi et al. [2015]. This article suggests a re-usable and unified definitions of AI for video games implementations. This feature could be a true technological progress for game developers. On one hand, the development time and cost could be reduced; on the other hand, every advances could be recorded and re-used in future games. This would bring a new diversity in games and give new experiences to players. However, I doubt this could be possible soon, or ever. Especially planners, but probably AI in general, are often very cases dependent in terms of design and implementation which makes difficult, almost impossible, to share codes between different projects. On top of that, each game studio has its own way of developing and coding games, using different tools and sometimes their own game engines and technologies. Hence, I think that a unified framework for AI in games could be difficult to create in the current state of the game industry. Anyway, this idea pleases me greatly and I hope this would be possible in the future.

7.5 Conclusion

I enjoyed conducting this project and discovering by myself several aspects of AI, specifically planners technologies. Working in a video game environment was interesting, and the supervision of my project was conducted smoothly. I don’t have any big regrets regarding the project and its conclusion. I am pretty happy about how it turned out, especially for the real-time performances, and wish to continue in the future.

Report written by —
Margaux Lussiana
Appendix A

NeoPacman, a student project in Python

This project called "NeoPacman" was part of a coursework in 2016 at ESME Sudria (France), and consisted of coding a small game in Python. Recreating a Pacman-like game implied coding the game-play specificities for the player side but also the AI requirements for the enemies. Those AI were coded using "if" statements to try to replicate the ghosts AI in the official version of Pacman. In that case, the use of such AI technology was sufficient as the enemy had to follow the player, but could be improved by the use of a simple FSM, or even a real-time planner. This example illustrates the various ways of coding AI depending on the situation and the illusion of smartness needed, but also highlights the weak ability of "if" statements to reproduce strong intelligence.

![Figure A.1: AI in NeoPacman game using if statements.](image)
Figure A.2: Game-play of NeoPacman

Project produced by —
Margaux Lussiana
Appendix B

Coursework – Search, Knowledge Representation, and Planning

In the scope of the Artificial Intelligence and Intelligent Agents (F29AI) at Heriot-Watt University, students were assigned a coursework around automated planning in groups of 2. I was grouped with Jonathan Meyer, and we had to model a space-ship domain using PDDL, as well as defining properties, objects, actions, goals, preconditions... The space-ship contains some personnel and equipment that need to be managed by the AI to fulfill goals, complex or simple. In the coursework statement, the professor imposed constraints and conditions to satisfy in any goals (such as charging robots). We implemented all the actions of the planner and decided their parameters, preconditions and effects on the world state. Finally we tested our planner with 2 crew sizes and 3 different goals.

We used the Fast Downward planner, which could handle our big domain and problem files. In order to find plans, we ran our planner with several configurations, such as the ”LAMA 2011 configuration” along with 4 other settings. Some of them failed for complex goals, but when they succeeded their plans offered a very high quality (both length and logic). Optimum plans were even reached for some goals-settings pairs.

PDDL is a powerful language to develop planners. We used it to write and define the domain and problem files of this coursework. Figure B.1 shows an example of the PDDL syntax for action definition. Figure B.2 provide a case with a complex goal and its associated plan founded by our planner.

Project produced by —
Margaux Lussiana, and Jonathan Meyer
Appendix B. Automated Planning

Figure B.1: One action definition using PDDL, from the coursework domain file

; ---------- MISSIONS -----------------------------

; First-contact mission
 (:action meet_alien
  :precondition
  (and
   (position ?capt ?planet)
   (position ?sec ?planet)
   (position ?a ?planet)
   (not (hasMet ?a))
   (not (injured ?capt))
   (not (injured ?sec))
   (not (= ?capt ?sec))
  )
  :effect
  (and
   (when
    (and
     (not (security_staff ?sec))
     (hostile ?planet)
    )
    (injured ?capt)
   )
   (hasMet ?a)
  )
)

---

Figure B.2: One complex goal and its associated plan, from coursework experiments

Goal

Associated plan
(FF configuration 2)
Appendix C

Risks Assessment

MACS Risk Assessment Form (Project)

Student: Margaux LUSSIANA (H00281112)

Project Title: Real-Time Automated Planner in Video Games

Supervisor: Dr. Stefano PADILLA

Risks:

<table>
<thead>
<tr>
<th>Risk</th>
<th>Present (give details) (tick if present)</th>
<th>Control Measures and/or Protection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard Office environment- includes purely software projects</td>
<td>Laptop, Unity software, mouse</td>
<td>Nothing</td>
</tr>
<tr>
<td>Unusual peripherals</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e.g. Robot, VR helmet, haptic device, etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unusual Output</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e.g. Laser, loud noises, flashing lights etc.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Other risks</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure C.1: Risks assessment form.
Appendix D

Search Algorithms

```csharp
foreach (Node successorNode in successors)
{
    if (!closedList.Contains(successorNode))
        continue;

    float try_gScore = successorNode.gScore + CustomMethods.dist(successorNode, currentNode);
    if (openList.Contains(successorNode))
        openList.Add(successorNode);
    else if (try_gScore >= successorNode.gScore)
    continue;

    successorNode.gScore = try_gScore;
    successorNode.fScore = successorNode.gScore + CustomMethods.heuristic_estimate(successorNode, goal);
    yield return new WaitForEndFrame();
}
```

**Figure D.1:** Code of the forward A* algorithm used for the planner.
Appendix D. Search Algorithms

Figure D.2: Code of the Best-First Search algorithm used for the planner.
Appendix E

Qualitative Evaluation - Questionnaire

This questionnaire sample is used for the qualitative evaluation with participants. Participants need to answer the following questions. The questionnaire contains nine questions and a score record table, on 3 pages. See figures E.1, E.2 and E.3.
REAL-TIME AUTOMATED PLANNER FOR VIDEO GAMES
QUESTIONNAIRE

Participant number : 0000

You accepted to take part in a project evaluation about artificial intelligence in video games. No personal information will be recorded.

You will be asked to play-test a mini game for as long as you wish (minimum of 30 seconds). In this game, you are going to play against a computer-driven bot. The main objective of the game is to kill the bot, but you are asked to pay attention to the behaviour of the bot while playing. Feel free to test any kind of actions and gameplay that could help you answer the following questions. Have fun!

Rules and controls:
• Control your moves with Z, Q, S, D keys. Control your view with the mouse pointer.
• Fire at the bot using Mouse Left click.
• Find ammo packages on the map to reload your weapon.
• You have 4 ammo.

QUESTION I)
Have you ever played video games before? If yes, have you ever played First Person Shooter games? (You can name them if you want)
Answer:

QUESTION II)
Did you find the prototype of this mini-game:

<table>
<thead>
<tr>
<th></th>
<th>Disagree</th>
<th>Somewhat disagree</th>
<th>Somewhat agree</th>
<th>Agree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fun</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interesting</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

QUESTION III)
Describe the way you play-tested this game. (E.G.: always trying to kill the bot, never moved, ...)
Answer:

---

FIGURE E.1: Questionnaire Qualitative Evaluation, page 1.
**QUESTION IV**
How would you describe the behaviour of the bot?
*Answer:*

**Figure E.2: Questionnaire Qualitative Evaluation, page 2.**

<table>
<thead>
<tr>
<th>QUESTION V) Qualify the different actions of the bot:</th>
<th>Answers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patrol</td>
<td></td>
</tr>
<tr>
<td>Fire</td>
<td></td>
</tr>
<tr>
<td>Find Ammo</td>
<td></td>
</tr>
<tr>
<td>Find Health Pack</td>
<td></td>
</tr>
<tr>
<td>Go to target</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>QUESTION VI) What do you think of:</th>
<th>Answers:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The bot's reactions</td>
<td></td>
</tr>
<tr>
<td>The visual features</td>
<td></td>
</tr>
<tr>
<td>The competitive aspect</td>
<td></td>
</tr>
<tr>
<td>The handling of the controls</td>
<td></td>
</tr>
</tbody>
</table>
**QUESTION VII)**
What kind of improvement would you like to make the bot more intelligent?

*Answer:*

**QUESTION VIII)**
After testing the second game, did you find any differences? Please describe them.

*Answer:*

**QUESTION IX)**
Other comments:

---

(This section can be left blank or filled several times.)

<table>
<thead>
<tr>
<th>Score</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AI</td>
<td></td>
</tr>
<tr>
<td>Player</td>
<td></td>
</tr>
</tbody>
</table>

*Play time: 0000 seconds*

Thank you for participating to this evaluation and survey!
Bibliography


