Human-like decision making for bots in mobile gaming

A case study in developing believable artificial intelligence for mobile games

Viktor Lundström

Viktor Lundström
Fall of 2016
Masters Thesis, 30 credits
Supervisor: Stefan Johansson
External Supervisor: Joel Jonasson
Master of Science in Computing Science and Engineering, 300 credits

Abstract

An AI framework for game AI of bots has been developed together with the company Level Eight for their upcoming game PROTOSTRIKE. The primary goal of the thesis is to implement a decision maker in the framework that exhibits human-like behaviours.

An overview and evaluation of existing decision techniques in the games industry have been done, and together with information gathered from the team at Level Eight, Utility AI was selected as most suitable for the decision-making system for the bots.

The implemented prototype of the AI framework is evaluated about the bots' ability to appear human-like in a multi-player environment. The results indicate that the lack of focus is the major behaviour that makes a bot not to act human-like. The test results have also been used to provide suggestions for future work and conclusion about the thesis as a whole.

Due to problems during testing, no statistical conclusions can be made. However, the comments made by the test subjects added insight how to improve the AI framework.

Acknowledgements

I would like to thank each and every one of the following people and machines for making this thesis possible:

- Level Eight for giving me a chance to explore game development.
- Joel Jonasson for mentorship and support at Level Eight.
- Stefan Johansson, Umeå University, for support.
- Susanne Kadelbash for making me strive to become a better person each and every day.
- Dave Mark for answering questions relating to Utility AI.
- Level Eight's coffee machine for making me into a manageable human being.

Contents

List of Figures

Chapter 1

Introduction

1.1 Background

Level Eight is an independent game development company located in Umeå, Sweden [1]. Level Eight is focused on the mobile platforms for Android and IOS. They have currently released six games, and they are in the process of developing a new action multiplayer game to be released this winter.

With their new game, Level Eight is in need of bots[1] in the game that will play on the servers when human players are not filling the servers on their own. This is done to ensure that the game experience does not falter when the number of players varies. To make the game experience equally good when a player is facing a bot as when the player is facing another player, the bots have to be equipped with artificially created humanoid behaviours to mimic a human player's actions based on some form of AI framework.

1.2 Thesis structure

This thesis starts off by explaining what purpose the thesis have and then goes on to describe what the prerequisites are in Chapter 3– 4 such as the general structure of PROTOSTRIKE and tools used for the development of PROTOSTRIKE.

After that, in Chapter 5 a general description of the main components used for developing the Game AI is given. Common techniques that are of use to answer the stated problems of the thesis are also presented.

In Chapter 6, a description of the developed prototype is outlined with details about the limitations of the design choices and development techniques.

Chapter 7 describes the conducted tests performed to evaluate the human-like behaviour of the bots using the prototype developed during this thesis.

Chapter 8 analyse the results from the previous chapter.

Finally, the thesis ends with Chapter 9 that reflects over the outcome, discusses the problems that have arisen, and speculates in what future work can be done on this subject.

[1] A bot is an AI system software that plays a character in the game like it was a human player.

Chapter 2

Problem description

2.1 Problem statement

The following section states the problem that is addressed by the thesis and requested to be solved by Level Eight.

2.1.1 User supplements

Currently, Level Eight is intensively working on their new game that primarily is a multiplayer shot-up game. Because this game will be launched on mobile devices, the users may lose connection to the game-server for different reasons. To amend this problem Level Eight wants to supplement the loss of real players during a game with bots to uphold the player experience.

Due to that, this is a new genre for them, the AI framework for the bots has to be developed from scratch. The AI framework will be running on the servers that host the game and therefore the AI framework's performance impact has a high priority to be as low as possible. The AI framework should also be easy to expand upon because the game will have more play modes and gadgets added to it during its lifetime. Therefore the bots have to be easy to upgrade with new behaviours needed to utilise new features. This puts further demands on the AI framework that the bots are built upon.

2.1.2 Humanoid interaction

The fact that the bots are intended to substitute human players puts a requirement on the bots to act in a similar fashion as its human counterpart would. Because of this, an investigation has to be made to identify key behaviours that make a bot, firstly, to be able to play the game and secondly, also to be able to act as if a human was playing it.

Not all humans play games the same way. Therefore, all bots should not play the game in a unified way. This induces that the AI framework has to be able to support alternating judgement when making decisions in behaviour so that not all decision outcomes are the same.

2.2 Purpose

The purpose of the thesis is to find solutions for the following problems:

1. Compare different standards and techniques common in the game industry for AI frameworks and to find out what advantages and disadvantages they have.
2. Select and implement a prototype of one of the AI frameworks.
3. Explore what behaviours are key factors for making a bot part of the game and act human like?
4. Evaluate performance weight against increased humanoid behaviour?
5. Evaluate if a bot is inseparable from its human counterpart by using the implemented AI framework prototype?

2.2.1 In-depth study

The primary in-depth study of this thesis surrounds standards and techniques for game AI, whereas their different strength and weaknesses will be compared so that the ones that suit Level Eight's game best is used. The main evaluations that will be addressed are the following:

1. Evaluate the performance impact of the different techniques and standards.
2. Evaluate maintenance of the different techniques and standards.
3. Evaluate expansion opportunities upon the different techniques and standards that are being evaluated.
4. How easy is it to maintain the code for the different techniques and standards?
5. Are the standards and techniques suitable for mobile game development?

2.2.2 Behaviours

Behaviours for the bots will be selected by gathering information from similar projects and feedback from Level Eight's game designers.

2.2.3 Benchmarking

Benchmarks will be conducted to ensure that the AI framework will not create a strain on the servers so that the player experience can be guaranteed.

2.2.4 Turing bot test

A Turing bot test will be undertaken to be able to say with some certainty if a bot can fool a human to think the bot is a human player. The Touring bot test will be based on previous research on the topic [2].

Chapter 3

Game engine

3.1 The usage of a game engine

The term game engine arose sometime in the middle of the 1990s. So what is a game engine? Well, that is a hard question to answer because there is no clear definition.

But some things that can be said about a game engine are that it is a piece of software that helps dividing game components into layers, and it is quite reusable at the same time. The game engine should also be able to, at least to some degree, act as a foundation for the development of a range of different types of games. How big this variety depends on how large the game engine is. This also means that not all functionalities in a game engine need to be used for every game that is created with the game engine[3].

3.2 Unity

Unity is a game engine that was first released 2005[4] and is today one of the key game engines on the market[5]. At Level Eight the main game engine used for the development is Unity. Some of the basic functionalities of this engine are briefly described in this section.

3.2.1 C#

The programming language that has the most support in Unity is C# but there is also the choice of developing in JavaScript. However, JavaScript has gotten less and less support during the recent years which has made it a less popular choice[6]. C# is a type-safe, object-oriented language that is developed by Microsoft[7].

3.2.2 Multiplatform development

Unity is a broad game engine that can generate games to a wide variety of platforms[5]. This also makes the engine a popular choice when a game is aimed to be distributed on many different platforms, and the development cost of particular platform deployment need to be minimised.

3.2.3 Visual editing

Unity uses graphical interfaces to make the development go faster for game studios[5]. However, both the code of the engine and the generated code can still be accessed by the

developer so that every aspect of the engine can be modified to fit the development style of each game being developed.

3.2.4 Game engine features

Unity has over the years added functionality so that it can be used to make a wide variety of games. These features include, but is not limited to, animation in 2D and 3D, in-game physics engine, AI support with systems for pathfinding and navigation meshes, and finally a grand amount of optimisation tools to make games work optimally on different devices[5].

3.2.5 Assets

Assets are representations of items that can be used in a project. Examples included, but not limited to, are audio files, pictures, and 3D models.

One of the features that have gotten most appraise in Unity is its Asset pipeline. The reason for this is the simplicity it gives when handling assets both when importing assets to a game, but also editing an asset so that when an update happens it is distributed throughout the architecture of the whole game.

Chapter 4

PROTOSTRIKE

4.1 Overview

PROTOSTRIKE is an arena based action *Player versus Player* (PVP) real-time multiplayer game developed for mobile devices, such as smartphones and tablets. The game is of the shooter type where the player has an eagle eye view of the character currently under control, which is more commonly known as a top-down style game. The controls in the game consist of a D-Pad to move a character and buttons to activate and use its abilities. A D-Pad is a standard control feature on game input devices that allows directional input. In this game, the weapon attacks and aiming are automatically performed for the player.

PVP is a game mode that surrounds humans facing other people in some game. Another mode that has close similarity to PVP is *Player Versus Environment* (PVE) where a human player is facing elements from the game itself.

Play styles may vary depending on the items and weapons the character is currently using. Each of these has their unique mechanic that can be either positive or negative depending on the situation.

A game normally takes three minutes and is played with up to eight players. If a player slot is not filled with a human or a human drops out while playing, it gets substituted by a bot.

4.2 Characters

A character has attributes that are unique regardless of what setup it has. The attributes in question are the health bar and the dash meter.

The health bar has a base value of 200, and when the character takes damage, the health bar depletes accordingly. The same thing happens when healing but instead the health bar recovers. The dash meter explained in Section 4.3.1 has two charges that can be used in rapid succession. When a dash is spent the character replenish that dash over time, the health bar also regenerates itself over time. However, the health bar's regenerative state can be interrupted by performing an offensive action or when the character is subject to such an action. After some time has passed with no interruptions, the character starts to rapidly regenerate health again.

All characters have some base attributes. A character can be improved in different ways by adding pieces of equipment and mods to it that adds passive attributes or different actions,

depending on the choices of pieces of equipment and mods for the character.

4.3 Abilities

4.3.1 Defensive actions and more

The Dash ability is used to a wide verity of tasks within PROTOSTRIKE and acts as the main defensive action in the game.

Ground Dash is done when a player is on the ground and uses the dash button. The Ground Dash makes the character rapidly traverse a distance in the direction it is facing.

Dash Shield is when a character performs any Dash it becomes invulnerable to most attacks, currently electrical based attacks are the only attacks characters are not invulnerable to when in dash shield.

High Ground Jump is when a character is standing close to a building on a spot that allows it to dash jump on top of that building which enables some benefits.

Gap Jump is performed when a character uses Dash close to a gap between two platforms and stands on a spot assigned a Gap Jump indicator. The effect of this is that the character jumps over the gap to land on the platform on the other side. Gap Jumps are unique in the way that when one is performed the character leaves combat mode immediately which can make health regeneration easier for instance.

Dash Sprinting is active when a player continues to hold down the dash button so that when the dash action is done, the character stow its weapons and enters a sprint mode where the character moves faster. In this mode, the character attack and the health and dash meters are not regenerated. When the character stops sprinting the regeneration starts again.

4.3.2 Attacking

As stated earlier the game handles the firing and aiming of the character, but a player can control some things when it comes to attacking. One is what target to fire at if there are multiple targets. There are also different firing modes for each weapon that the player can choose from during play.

Auto Attack is the weakest of all attacks. This attack starts when the character gets close enough to become targeted. The Auto Attack stops when there are no enemies in line of sight, i.e. the enemies are too far away, or they are in a place that the Auto Attack cannot reach such as the enemy being behind cover.

Rapid Attack is done when a player taps the attack button in rapid succession. As the name states, the attack is a bit faster than the Auto Attack and it is mainly used to finish the enemies off when they have low health.

Aimed Attack is activated when a player holds down the attack button for a moment charging up the attack and when the charge is full the player lets go of the button to deliver an Aimed Attack. Aimed Attack has a high damage output and also hit enemies that have positions that regular attacks cannot normally hit.

Powershot Attack is the strongest attack in the game and it requires the player to find ammunition before it can be used. Powershot Attacks are done by holding the Powershot button and just as the Aimed Attack it has charge functionality. While charging a Powershot attack it can be aborted by the use of tit he dash function. There are two modes of Powershot

where one of them has a stronger effect but takes a longer time to charge. The other one can be activated by simply tapping the Powershot button, and it still takes some time charging before it fires and gives less of an effect than the strong version.

A character can if there are multiple enemies in sight change their target by pressing the switch target button. This overrides the automatic selection of target where the closest target always is chosen. Using the switch target button a new target is selected by increasing distance from the player; the second closest enemy is chosen and so forth. If a player uses an active action on a target, the target will not change if another enemy becomes the closest enemy unless an active choice is made to switch or if the selected target moves out of range.

4.4 Positioning

In PROTOSTRIKE, two of the main focuses are position and map awareness, i.e. positioning in comparison to where the other players are located is the key! A big advantage is achieved by taking the high ground by getting on top of a building. This affects the game in that characters trying to hit the one on top of the building will miss all their shots unless they perform an Aimed Attack. On the other side of the situation, the character on top of the build can use all types of attacks to hit the other characters within a line of sight.

4.5 Objects

Like most arena based shooter games there are pickups dispersed around the maps, and they are used to improve the characters instantly in some way when picked up. These pickups are briefly described below.

4.5.1 Health packs

Health packs increase the character's life when picked up. There are normally spots on the maps that spawn health packs but a player can also gain health from health packs that are spawn when another character is killed, however, these types of health packs are somewhat smaller in the amount they heal.

When a character picks up a health pack, and it fills a character's health above standard health (200) then all health above 200 become Overhealth. The difference between normal health and Overhealth is that Overhealth decreases over time until the character has a life count of 200 or less. The maximum Overhealth is 100 bringing the total maximum health up to 300.

4.5.2 Weapons

In PROTOSTRIKE there is a wide variety of weapons. Each weapon has some differences when it comes to the form of attack. For instance, an assault rifle shoots fast and in a straight line while a shotgun fires with a slower rate but has a wide area it hits when firing.

The special attacks mentioned can also change depending on the weapon so a completely different approach may be needed to be competitive. This means that depending on what type of weapon a player is using on their character, the play style they use needs to be different to maximise the effectiveness of the current weapon.

4.5.3 Ammunition packs

Ammunition packs are needed to be able to perform a PowerShot Attack. These pickups are scattered on key locations of each map to create fun encounters. As a PowerShot Attack is the most powerful attack in the game these ammunition pack pickups are of high value to the players.

4.5.4 Dash Refill

A Dash refill pickup gives the character an instant refill of a dash when picked up. To be able to get a refill the character needs to have at least one of its dashes used up at that point.

4.5.5 Loot Crates

Loot Crates randomly appear on the map during the play session. A Loot Crate becomes activated when a character is not in combat mode and stays near it for a duration of time. An activation phase of a Loot Crate can be cancelled if the character doing the activation enters combat mode or by moving away from the loot crate or doing any active action. When a Loot Crate becomes activated it spawns two random pickups. Loot Crates are chest like objects that opens when they are activated.

4.6 Game mode

Currently, Level Eight is only developing the game for one game mode, and that is the common type of gameplay called *Free For All* (FFA). This type of deathmatch has the following rules:

- All players in a game consider all others as enemies.
- The player with the highest score wins the game.
- The score is gained by the following actions: killing an enemy, doing damage to an enemy, and bonus scores from, e.g., kills in a row without dying.

Level Eight has plans to expand the different modes to play PROTOSTRIKE in the future but is not actively working on it yet.

Chapter 5

Standards and techniques for game AI

In the beginning, there was chess. That is to say at the beginning of *Artificial Intelligence* (AI) in games. The first article about how a computer would be able to play the game of chess was written in 1949[8]. Since then not only have AI proven to be able to beat humans at games like chess but they have also started to replicate humanoid behaviours to different degrees. Humanoid behaviours have become something to expect when playing video games these days and therefore it increases what a consumer demands of the AI in games even further.

This has brought high demands on the implementation of the AI framework as well as the underlying decision maker. Some of these requirements are the following[9]:

- Be able to handle massive amounts of input data and then decide from this what behaviour to use for the situation.
- Have humanoid behaviours and act in a logical way.
- Be able to show complex and emergent behaviours.

With these demands, an AI framework still needs to be easy to implement, support, and expand for further needs of a game studio.

Currently, the making of a game AI is diverse. There exist so many game types and different frameworks that all have their benefits and shortcomings that there is not anything that can be called a standard for how to create a game AI. We begin this chapter by in the first section give an overview of the most common techniques for decision making.

5.1 Techniques for decision making

In the game's AI, decision making is one of the most important parts for achieving AI. The decision maker tells a bot what to do next. This can vary from game to game how complex the decisions have to be. The decision made can be everything from a small, simple decision to large chained decisions that have many steps in them[10, Chapter 1.2.2].

5.1.1 Finite State Machines

The most common and simplest techniques to handle decisions are based on a finite state machine. A *Finite State Machine* or an FSM are used in many parts of programming[11].

An FSM consists of a set of states and a set of transitions making it possible to go from one state to another one. An example of a transition graph is shown in Figure 1. A transition connects two states but only one way so that if the FSM is in a state that can transit to another state, it will do so if the transition requirements are meet. Those requirements can be internal like how much health a character has, or it can be external like how big of a threat it is facing.

This technique offers lots of flexibility but has the downside of producing a lot of method calls. In most implementations of a flexible FSM for games, it only requires the memory to hold $O(1)$ state and $O(m)$, in time. Where m is the amount of transitions per state. However, for both the states and transitions, other methods are usually called and takes most of the time in an FSM[10, Chapter 5.3.1-7].

There exist also hard coded FSM that was a common standard earlier which only uses enumerated values to check what state it was in. This implementation is faster than the flexible, but it is extremely hard to maintain if changes or expansions are needed for the FSM. The biggest problem seems to be that it is usually integrated into the main game, so that if changes are done the whole game has to be rebuild which can take hours for a big game project[10, Chapter 5.3.8].

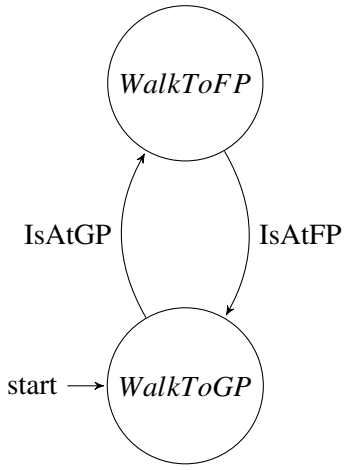


Figure 1: A transition graph of a Finite state machine describing a bot walking between a flag pole and a guard post.

An FSM is conceptually easy to understand but the main problem with them are often to add new states/behaviours which when completed should go back to its previous state and continue to do whatever is in that state. As an example, we have a bot that should walk between a guard post and a flagpole. The states needed for this is one for the state of walking towards the flagpole and one for walking towards the guard post. The transitions between them would check if the bot is walking towards the flagpole and if it is at the flagpole it would tell it to transition to the state of walking towards the guard post and the other would check for the opposite. The resulting transition graph of the FSM is shown in Figure 1.

If we were to add a new state that handled, for instance: the bot gets a call and should stop what it is doing and talk in its phone and when done talking return to its previous task. This would in this small FSM require adding two new states for the phone call and four transitions to be able to handle this, see Figure 2. As one can see, returning to a previous state when getting interrupted can expand the size of an FSM quite a lot.

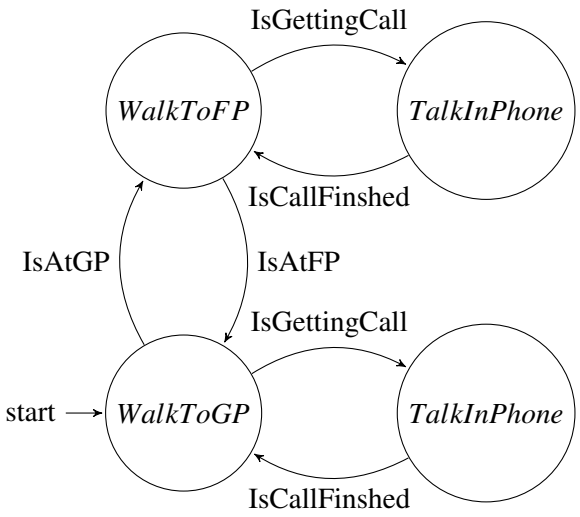


Figure 2: A transition graph of a *Finite state machine* describing a bot walking between a flag pole and a guard post with the ability to get a phone calls.

5.1.2 Hierarchical State Machines

There is a technique that solves most of the problems with FSM. The solution is called a *Hierarchical Finite State Machine* (HFSM) where the main advantage being each state can be a complete FSM of itself[12]. An HFSM still has much in common with an FSM but makes it somewhat easier to maintain and reduces in some cases a lot of performance problems. HFSM requires $O(n)$ in memory where n is layers of an HFSM and $O(nt)$ in time whereas t stands for transitions[10, Chapter 5.3.9]. See Figure 3 about how HFSM solves the problem related to Figure 2.

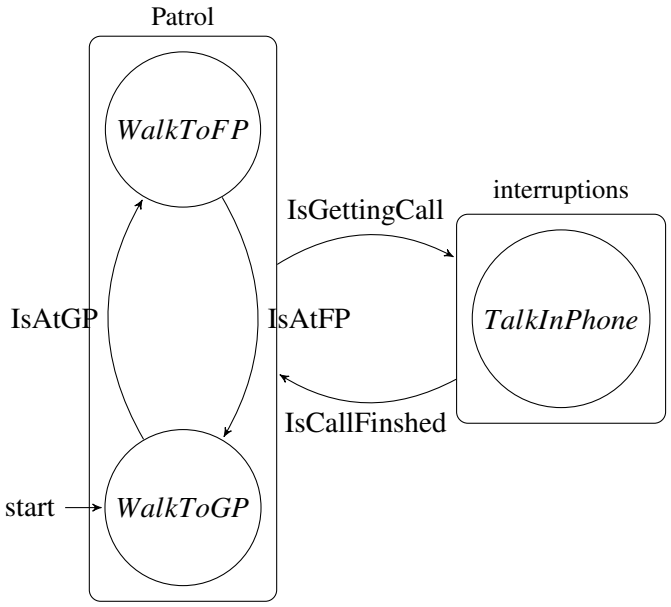


Figure 3: A transition graph of a *Hierarchical Finite State Machine* describing a bot walking between a flag pole and a guard post with the ability to get phone calls.

5.1.3 Decision Trees

A *decision tree* is one of the easiest decision-making technique there is[13]. There are many ways this technique has been expanded upon but right now let us describe its basic form.

The way a *decision tree* works is by letting each non-leaf-node be a decision while each leaf-node be an action or a behaviour it should start. *Decision trees* are usually used in decision making that requires speed and are still quite simple. A decision in a *decision tree* is usually made by checking a single value to move further along the tree.

In a decision, there are usually no logical checks such as AND or OR because they are implicitly defined by the tree itself. This is exemplified by the example in Figure 4. If a decision is made from checking if a character is hungry and if it has food the then this would be represented in a *decision tree* by first checking if it has the value of hungry, and if yes it would step down in the branch that has the value-hungry equals true. The next decision in that branch would be to check if the character has found food or not which tells which branch it should choose next.

Decision trees can be built in a non-binary fashion making each decision have more than two possible outcomes, but making a *decision tree* non-binary has shown to not have that big of an impact[10, Chapter 5.2.3]. It depends on what decisions are required to tell the programmers what design they have to use on their decision tree.

The performance of a *decision tree* is quite fast, especially if the tree is balanced then the algorithm is $O(log_2 n)$, where n is the total number of decisions in the tree. But if the tree is unbalanced the worst case become $O(n)$ [10, Chapter 5.2.7-8]. There are ways to make the performance less likely to get the worst case by setting more common actions higher up in the tree, but if this is not possible it is on a case by case basis. The problem with *decision trees* becomes apparent when new behaviours or values are introduced to an already partly developed decision tree, which is likely to happen during the development of a bot in one way or another.

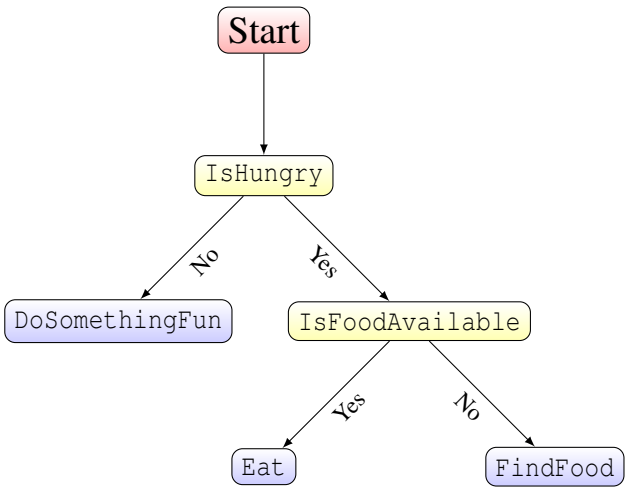


Figure 4: An example of a *decision tree*.

5.1.4 Behaviour Trees

Behaviour trees reuse many concepts and techniques stated earlier in this chapter and improve upon them by combining some of them. *Behaviour trees* became more popular to use after it was announced that the game Halo 2 used it[14].

Behaviour trees are similar to HFSM but the functionality of a state is replaced with a task. A task can be described as a small action if it can get a value or activate an animation[10, Chapter 5.4]. The task is arranged in a similar fashion as the nodes of a *decision tree* with the difference that actions happens not only on the leaf nodes but rather on all the nodes.

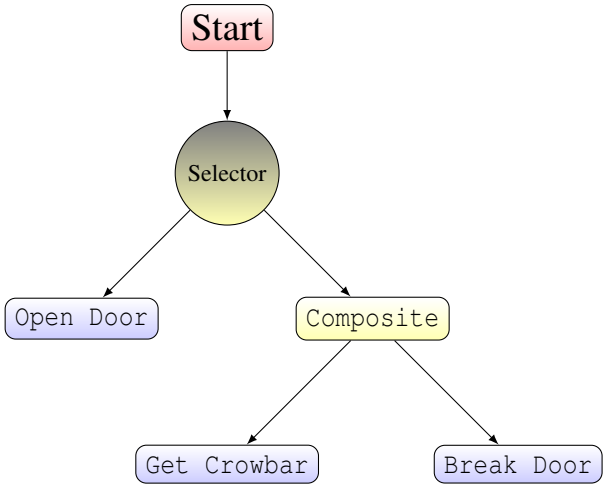


Figure 5: An example of a *behavior tree*.

With this structure, a chain of tasks can create a greater behaviour than the small actions each task does on its own. The general guideline is to break the task down into the smallest possible action set and still to be useful when chaining them. A task always tries to perform an action, and when it is done, depending on the action, the task returns a status value that often tells how the traversal shall continue from that point forward.

Tasks can be divided into three sub-categories: Conditions, Actions, and Composites. Conditions can be described as test statements on some value. Actions changes some state in the game, for example, increasing the aggressiveness value of a player as the Action Task handles rage. And last we have the Composite task. The Composite task is used to execute its child nodes and by doing that a *behaviour tree* can accomplish some very profound behaviours for an AI. How the Composite task chooses what child nodes will be executed depends on the way the composite is implemented.

There are of course many ways to implement tasks but all are depending on what kind of behaviours the developer is planning to create, therefore will we not go into further depth on implementations.

Because of each *behaviour tree* looks different, talking about its performance more than what has already been said about other trees in this chapter is hard. A *behaviour tree* is quite modular and even well-known for its intuitive way of describing behaviours to a non-programmer. Therefore it is loved by game studios when designing bots, especially since all parts of a game studio can easily understand how each sub-behaviour acts [10, Chapter 5.4]. Maintaining a Behaviour Tree benefits greatly from the fact that all tasks can be divided

into three categories as stated earlier, but also that most of the known implementations of *behaviour trees* use the same interfaces for all tasks.

Behaviour trees use what is called a reactive design where the AI tends to try things and makes its decisions from things it has gotten signals from. This is good for fast phasing games where situations change quite often. On the other hand, this is bad in more strategic games where many moves should be planned into the future without real feedback. This problem has been solved to varying degrees by adding other functionalities or giving the *behaviour tree* made up responses.

5.1.5 Goal-Oriented Action Planners

Just as the previous decision-making techniques are more of reactive behavioural styles *Goal-Oriented Action Planner* (GOAP) tends to be more of a planning structure. GOAP became quite popular after the studio behind the game F.E.A.R released an article about how they had solved the AI of their game with FSM and mainly GOAP [15]. GOAP general way of thinking is to give a bot one or more goals then give it some actions and let it figure out how to achieve its goal/goals with its actions.

A goal in a GOAP can be seen as the main motives for a bot to do what it does. A bot with GOAP can as stated earlier choose from one or many goals to pursue, goals are frequently prioritised in some way. This prioritisation can in real-time be changed if as an example the goal of being healthy increases in priority when the health goes down. Other goals might change by the flow of time like becoming sleepier the longer the bot is not sleeping. When some time has passed while sleeping the purpose of being rested will instead have the highest priority.

Actions are behaviours that fulfil a goal or part of a goal. By stringing actions together, a goal may be achieved but each action usually has a cost, and often many different action sets can achieve a goal, but they usually have different costs. This difference determines what set of action or plan to use, and that is where the name of this technique comes from. Often an action has some prerequisite that needs to be fulfilled to be able to do them and often results in that another action is needed before the first evaluated action can be done. An example of a set of actions in an action game can be: draw the weapon, aim the weapon, and shoot weapon.

To find the best set of actions to fulfil a goal it is common to either use depth first search or A* search, which is mostly known as a pathfinding algorithm[16]. The big difference is that in pathfinding no restriction is usually set to the depth of the search while in a GOAP search their needs to be a max limit to the depth. This is because there will almost always be more actions to do after a previous action if it is given a choice.

The performance in time is $O(n^d)$ where n is the amount of actions that are available and d is the max depth set for the search [10, Chapter 5.7.6]. GOAP is easy to implement because of its modular components, and it lets the final behaviours emerge from the actions given the bot. This can be both good and bad as the bots behaviour is easy to get up and running but might be hard to figure out how to change if something is a bit off. This is due to properties that emergent systems have[17]. The studio behind the rebooted game series Tomb Raider uses GOAP, and their main advice is to invest in debugging support [18]. On the positive side, there are no problems with adding actions and goals to the system as it does not add that much to the overall complexity.

5.1.6 Utility AI

While most of the technique previously discussed are based on logical expressions, *Utility AI* is not. Utility theory has been around much longer than AI has been used for games and has mainly been used in economics[19]. It has recently started popping up as an alternative in games with AI. To describe *Utility AI*, in short, the idea is that every possible action can be described with a single uniformed value that tells the actions usefulness [20]. So by calculating each actions usefulness value, the AI knows what action is best to do in its current state of being. The usefulness value is often called utility.

Calculating different actions utility can be hard because they usually get their value from different parameters. For instance, the action eating and the action sleeping does not have anything to do with one another, but they are almost of equal importance. So when calculating their utility, hunger and tiredness will each be factored such that if the bot is more hungry than tired, then it will eat and vice versa.

The key part of *Utility AI* is that the utility can be calculated from simple values added together with some mathematical expression that balances the impact of what the values mean to what an action is useful for. Kevin Dill and Dave Mark described how to add together values to create utility values beautifully at a GDC conference [21].

The positive thing with using a mathematical expression for describing behaviour choices is that it is easy to decipher why an AI does something and therefore easy to change depending on the input that is given about an AI so that it can fit its role in an optimal way.

Performance Wise there is not that much to go on because of the fact that this type of decision-making framework is somewhat cutting edge and is, therefore, more keen to have information about it hidden as trade secrets. A thing know about *Utility AI* is that it is fast to generate new bots with different behaviours. This is shown at GDC wherein the presentation the speaker demonstrates how it takes 7 minutes to create the AI behaviour for a new character[22].

5.2 Final thoughts on decision making

While many of the decision-making patterns described in the previous section have been critical for the advancement of AI in games, not all of them are practical for the high demands of today's games.

Among these FSM and *Decision Trees* are today not only too simplistic to create advanced behaviour decisions, but the main reason they are not suitable for the current state of game AI development is that they are hard to expand and maintain.

Hierarchical finite state machine solves some of the expansion problems with FSM but it can still be somewhat hard to maintain in large complex AI systems. For simple systems, the HFSM is an entirely valid way to solve decision making.

Behaviour trees are a bit better than HFSM in that they use a structure that makes it even easier to expand but it has the same problem with complexity and maintenance as HFSM.

GOAPs are a step in the right direction according to AI programmers community in the game industry, as it deals with the earlier problems about complexity and maintenance by dividing them according to the motives of a bot. The downside of GOAP is that it is harder to understand conceptually and implement, especially when bot behaviours come from emergent properties.

Utility AI is one of the newest ways to create decision making and depends heavily on math to compute the valid decision. Since it is one of the newest techniques in the field to be used for digital games, it is not as well documented as the earlier techniques and that can be a big disadvantage.

As a conclusion, one can see that there are many techniques to handle decision-making all of whom have their strengths and weaknesses. This also means that not all techniques are suitable for all situations.

The technique that this thesis will continue with is the decision-making technique Utility AI. This is because it has been shown to be able to quickly generate new behaviours which are critical to being able to expand the AI for a game.

5.3 Pathfinding

One of the most common things for bots is to have instructions about how to move or, in particular, find the (shortest) way from one position to another. All the way since Pong, bots have had movement as a core function[10, Chapter 3]. To make a bot move is however not enough! They need to have a goal with their movement, if it is getting somewhere or getting from something is not important. Indifference to popular belief, the journey is not the goal when travelling. It is, however, important to know all the steps of the voyage.

This is where pathfinding comes into play. Pathfinding is a problem that has had different expression throughout of human history like the travelling salesman problem and others like it [23]. Often have these problems been represented with graphs where the nodes represent possible goals, or current positions and edges represent the distance/effort to get from one node to another one.

Pathfinding in games still use these principles to varying degrees and have developed many ways to calculate the fastest way to get from one position to the goal position.

World representation is of great importance as it dictates how well a decision can be made since all information about how it is possible to move around comes from the world representation. There are many ways to represent the world with different search spaces [24] ranging from grids, graphs, and navigation meshes. Each of these has their different advantages and disadvantages.

Regardless of the search space representation, there is still need to search through it to find a path from the current position represented in the search space and the goal position that also has a representation in the search area. There are many ways to achieve this trough algorithms such as A*, Dijkstra's Algorithm [16], and much more.

5.4 Knowledge Representation

Just as there are needs to have representations of how it is possible to move in a game, there are also needs to have representations about in what state other things in the world are in that can have an effect on how a bot acts.

Building this representation is one of the most difficult things to do when designing an AI framework[10, Chapter 3]. The easiest part is to get information about what a bot can directly access using techniques like ray casting or spatial awareness where everything within an area or a volume the bot can query. Exchanging information between bots is an-

other matter completely since this requires protocols of some sort to enable communication between the bots.

The representations should uphold and honour the following demands as good as possible for them to be a valid choice[25]:

- Easy to integrate and maintain – If this is not upheld future development might become impossible for the bot to ”understand”.
- Flexibility – So that many types of information can fit in the same representation. This also helps with the demand above.
- Minimise iteration time – This can be hard depending on the type of information and how it is gathered.

Information is in many cases uninformed from several interfaces which all have some base elements such as entity, membership, position, and classification. This is done to make knowledge parsing to be as smooth as possible.

5.5 Humanoid behaviour

To construct a computer that can reason and act like a human is one of the primary goals in AI research. Alan Turing wrote in *Computing Machinery and Intelligence* [26] about The ”Imitation Game”, where a computer should try to fool a human with its answers to text questions; if the human thought the answers came from another human the computer would have succeeded in the test. Today, this type of test is called a Turing Test[27].

There have been many objections to what Turing wrote on artificially intelligent, but for the purpose of game AI this has no direct effect as the reason for having bots is not to achieve real AI but making bots act human. Philip Hingston formulated a test for bots based on Turing's imitation game, but whereas a success in the imitation game would prove intelligence, Philip Hingston's *A Turing Test for Computer Game Bots*[2] would only show that the bot was Anthropomorphic[1].

Philip Hingston used his test within the competition *The 2K BotPrize* to see who could create the most humanoid bot. The game that was used for this competition was *Unreal Tournament 2004*, a well known multiplayer FPS game, where FPS stands for *First-Person Shooter* which is an action game type that allows the player to experience the game world (often in 3D) from a first-person perspective.

After the competition was completed, some conclusions could be made for what distinguished a bot from a human. The following characteristics were give-aways for bots not being humans:

- Missing behaviours (Common behaviours that humans had but not the bots displayed)
- Stupid behaviours (Getting stuck or doing illogical decisions of what behaviour to use for some situations)
- Low aggression (Human players displayed a far more aggressive behaviour than the bots did)

[1] Anthropomorphic means having human characteristics.

Human players were spotted by the following:

- High aggression (The opposite of the bots)
- Adapting to the opponent (Changing behaviour depending on the responses from enemies)
- Good tactics (Long time formulated plans of attack)

Humans often add a personality to objects depending on the way they move; this is shown in the classical psychological experiment *"An experimental study of apparent behaviour"* by Fritz Heider and Marianne Simmel in 1944[28]. The experiment was conducted by showing a movie of two triangles and one circle and asking the people that saw the movie one question: What did you just see? The experiment was originally conducted on 34 participants. 32 participants described the two triangles and the circle with different personality traits and the remaining two described shapes with bird-like behaviour. The experiment is well known in psychology as showing humans ease to anthropomorphism objects. Choosing different movement patterns can therefore just as in the experiment create the sense of personality traits.

5.6 Tactics

Tactical behaviours that are logical are key foundations to make a bot believable. As an example, a bot uses its pathfinding to get near its enemy to hurt it. If that enemy has a gun while the bot has a knife, just walking up to it might be a bad tactic.

Today's game AIs are held to a higher standard than doing something as stupid as the previous example. That is why tactical behaviours are needed. By adding information from the search space about the area around the enemy, behaviours can be implemented that gives the bot a higher rate of success and therefore seems smarter.

By changing behaviours by small amounts can often make a player associate a bot with different personalities. The simplest example of this is to look at the famous game Pac-Man, where all the ghosts have different chase behaviours which can be interpreted as the ghosts have different personalities[29].

5.6.1 Ambition

Bots often have some goal with its existence and behaviour is therefore implemented to give the incentive to do actions that put bots closer to one of its goals. GOAP is highly proficient in doing this type of heuristics.

5.6.2 Self preservation

Many of today's games are centred around the action in some way. To create a believable bot, they have to "care" when they are hurt. This means there have to be behaviours that mimic the feeling of that a bot cares when it is getting hurt. Often this is implemented by searching for cover, giving signs that it feels hurt by animations or sound, or by trying to run away.

5.6.3 Curiosity

What does a bot do when it does not get any input that has an effect on its ambitions? Well, this state of being is solved with either going idle and not doing anything until something that is of importance for the bot happens. Another thing the bot can be implemented to do is search for something that has an effect on its ambitions, and this can be done in most games by moving around the map in some fashion.

Chapter 6

AI prototype in PROTOSTRIKE

During the thesis, a prototype of an AI framework has been developed, where the underlying decision maker is implemented using Utility AI. This chapter explains the different parts of the framework with the focus on Utility AI and how it is integrated into the game. For a UML description of the framework see Appendix A.

6.1 Utility AI

Utility AI makes decisions about what type of behaviour Should be done in different situations. The behaviours can be anything from when a bot should jump up on a ledge to which enemy it should shoot at first. The Utility AI works by giving a bot a set of actions, where each action has at least one function associated with it, called an *Axis*. For every Axis in action, an evaluation of its utility value between zero and one is done. Once all Axis for a particular action has been evaluated they get multiplied together to a score. This score is then added with a weighted score to produce the resulting action score that shows how important an action is, for details see Section 6.1.3. The action score gives a weight on how good the outcome of that action is at the current state/time. Once all actions have been evaluated, there are two choices on how to pick the action to be instantiated. Either pick the one with the highest score or use weighted randomization where the weight of each action depends on their action score.

Actions come in *Action Packets* that are put together from the type of abilities the actions have and what kind of bots will use them. The same goes for Axis which are packed in Axis Packets. It is mainly done this way to categorise the Axis and Actions quickly.

The implemented Utility AI prototype can be seen as four distinct systems that have their tasks that are all critical for making a decision. These systems are Sensor, Axis, Actions, and Utility Frame, which are described in the following sections. See Appendix A for the UML description of the Utility AI prototype, which is described below.

6.1.1 Sensor

The Sensor system has to be implemented differently for each game that uses the AI framework. The main function of the Sensor system is to gather information about the state of the game that the bots use for evaluations. This can be completely different from game to game. Like how far should the bot be able to ”see”? Or how much should the bot be able to remember about other players and for how long? Many of these questions are related to

the game type and what kind of actions one wants the bot to be able to do.

The current implementation of the Sensor system in the PROTOSTRIKE takes into account all things interactable within a predefined distance from the bot. The information the bot can evaluate should mimic how much a real player can see on the screen when playing. The Sensor uses a snapshot principle which means that it only gathers information when a predefined period has passed by.

As an example, the Sensor in PROTOSTRIKE has a grid of positions where the bot is in the middle, whereas the bot can check if it wants to move to any of those positions. A list is made for each interactable item which is currently ”seen” by the bot so that it can evaluate if it wants to interact with anyone of them.

This information is stored in the snapshot and is used for two things: The first one is to generate concern values that are used as input for the Axis functions so that they can generate their current utility value. The second reason is to generate Context objects that are passed from the Sensor so that actions that need a target gets evaluated on every target which is valid for this action. For example, if the action ”Shot bad guy” were to exist then the Sensor would have a list of bad guys in the snapshot of the bot can periphery. The generated Context object for evaluating that action would then include the list of bad guys. This means that some actions might appear many times during the evaluation of which action is currently best to do like shooting bad guy A with one score or killing bad guy B with another.

6.1.2 Axis and concerns

All concern values generated by the Sensor of the bot are converted to a floating point number between 0 and one by the Axis. How important the inputs depend on the concern's conversion function in the respective Axis. The conversion function can be any mathematical or logical function that takes one numeric variable as input.

The currently implemented types are Linear, Quadratic, Logarithmic, and Sinus functions, and a Boolean based function. All functions except the boolean function have been standardized so that they all uses most of the same six parameters: m, k, b, and c to define (6.1) – (6.3), and *maxValue* and *minValue*. The algorithm of the boolean function is shown in Algorithm 1 and the remaining functions are:

$$Y(X) = m * (X - c)^k + b, \tag{6.1}$$

$$Y(X) = (\log_k(X/(1-X)) + (10 * b))/(10/m), \text{ and} \tag{6.2}$$

$$Y(X) = \sin((X - c) * m * (2 * \pi)) * (k/2) + (b + 0.5), \tag{6.3}$$

where m is used to set the slope of the curve or in sinus equation the frequency of the wave. k is used for the vertical size of the curve in the logarithmic and sinus equation while in the linear it should always be a one or else it is no longer linear. b moves the curve vertically, and m moves it horizontally. The two remaining parameters convert the input value to a number between zero and one from a *maxValue* parameter and a *minValue* parameter of what is considered acceptable boundaries. X is the incoming concern value that has gone through the conversion and if needed clamping. And finally, Y is the Utility value of the Axis. For visual support please see Figure 6– 8 [30].

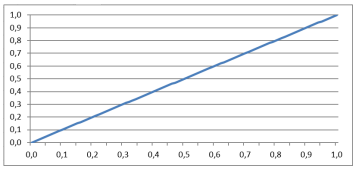


Figure 6: Linear curve with the following parameters: $m = 1, k = 1, b = 0$, and $c = 0$

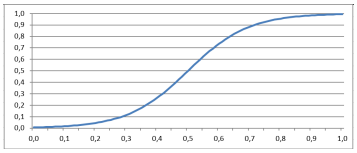


Figure 7: Logarithmic curve with the following parameters: $m = 10, k = 1, b = 0$, and $c = 0,5$

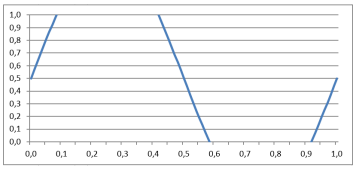


Figure 8: Sinus curve with the following parameters: $m = 1, k = 2, b = 0$, and $c = 0$

If the input is outside one of the boundaries then it gets clamped down to the nearest boundary so that it becomes either a one or a zero. The boolean function needs a variable flag to tell if a value over some limit should return a one or a zero, see Algorithm 1. The output is also always clamped between zero and one.

```
   input  : Context based value V for Concern
   output: Utility of concern
 1 V = (V - MinValue) / (MaxValue - MinValue);
 2 if V > BreakPoint then
 3     if isHighPositive then
 4         return 1.0;
 5     else
 6         return 0.0;
 7     end
 8 else
 9     if isHighPositive then
10         return 0.0;
11     else
12         return 1.0;
13     end
14 end
```

Algorithm 1: A quite simple Boolean function that creates a utility for a concern.

All Axis are stored in a container which is of the singleton type. This container is only used to store the Axis and give references to the Axis objects on request when a new action is created during the startup phase.

6.1.3 Actions

All actions made in PROTOSTRIKE can be divided into two categories: movement or attack. The move actions are then divided by the desired effect of a change like moving close to an enemy or moving to get out of sight for the enemies. Attack actions are different attacks that are possible for a player to do. Currently, there are four different types of attacks (Normal attack, Rapid attack, Aimed attack, and Power shot) which are described in detail in Chapter 4.

Each action has a reference to each associated Axis that are used to compute the action score. Some actions may share the same Axis whereas each has a reference to the same Axis object.

The final score (*FinalScore*) of an action is computed from the individual scores S_i of each Axis as:

$$FinalScore = SumScore + (MakeUpValue * SumScore), \tag{6.4}$$

where

$$SumScore = \prod_{i}^{nrOfAxis} S_i, \tag{6.5}$$

$$MakeUpValue = (1 - SumScore) * ModificationFactor, \text{ and} \tag{6.6}$$

$$ModificationFactor = 1 - (1/nrOfAxis). \tag{6.7}$$

The equations (6.6)–(6.7) ensure that an action with many Axis does not get punished for it[22].

The final step is to multiply *FinalScore* with the Rank that the Action is given and this is the score of an action. The Rank is a positive integer that tells the overall importance of an action independent of the situation.

6.1.4 Action Packets

All actions are stored in JSON files that are called Action Packets before the bot is created. Depending on which type of bot is wanted, different Action Packets are used to generate that bots actions.

When a bot is given an Action Packet, it uses a factory pattern to generate the actions that are described in the Action packet.

6.1.5 Utility frame

The Utility frame is what keeps the AI framework together and works as a controller. After it has been instantiated it takes in data paths to generate Axis and Actions from text-based formatted files so-called Axis Packets and Action Packets, in this case in *JSON* format. It also has an instance of the Sensor system discussed above. The Utility Frame is in charge of going through all actions and gives them the right information from the Sensor to generate each action's final utility for each target or targets. After that, a final decision on all the action's scores is generated by the Utility frame. See Figure 9 for a simple example of how the steps in the Utility frame can look like.

6.2 Behaviours

A behaviour corresponds to an action, so when an action has been chosen its corresponding behaviour is started. Just as a big part of the Sensor system needs custom development so does almost all behaviours.

When the bot wants to start a new behaviour, it asks the decision frame to give it the best action currently. The decision frame then evaluates each action with the context generated from the Sensor's current snapshot and what target the current action in evaluation has. Once all actions have been evaluated the framework returns the best action to perform.

All behaviours that are implemented in PROTOSTRIKE uses the functions of the weapon systems and movement system. The weapon systems can start different fire actions and change which target the bot is currently aiming at. The movement system can start all behaviours that are connected to movement actions.

The behaviour components have all been to some extent designed by personnel at Level eight before this thesis project started.

6.3 System Overview

When designing the AI framework some fundamental design choices have been done. These are briefly discussed in this section.

6.3.1 Unity: *NavMesh*

Unity has a built-in pathfinding system called *NavMesh*. The *NavMesh* uses navigation meshes and has many built-in features that are used by the movement system part of this prototype. The movement of the bots and the knowledge gathering about different positions of the game world uses the *NavMesh* as a tool.

To make the *NavMesh* system understand how dash jumps work, special constructs on the game maps had to be created to trigger the bots dash at the right time. The dash jump constructs are quite simple and contain a start position and an end position connected by a *Off-Mesh Link*, and each possible dash jump has one of these constructs. If a bot using the *NavMesh* stands on a start position of a dash jump and its next position to move toward is the correlating end position a dash jump is initiated.

6.3.2 Data-driven implementation

A key feature of the prototype is that it should be easy to customise. This means that actions and Axis that are bot dependent and often altered should, therefore, be easy to change. The actions and Axis have because of this been implemented as data-driven structures where all the data is stored in JSON format. By separating it this way people developing the bots without much programming experience can still change how a bot behaves.

6.3.3 Performance testing

During the whole implementation of the Prototype *Unity,* built-in profiler tool has been used to make sure that the prototype stayed within acceptable performance demands. Adding more behaviours has not decreased the performance in any significant way during the whole process. The big jumps in performance have been in direct correlation to implementation standards of different components and their effectiveness.

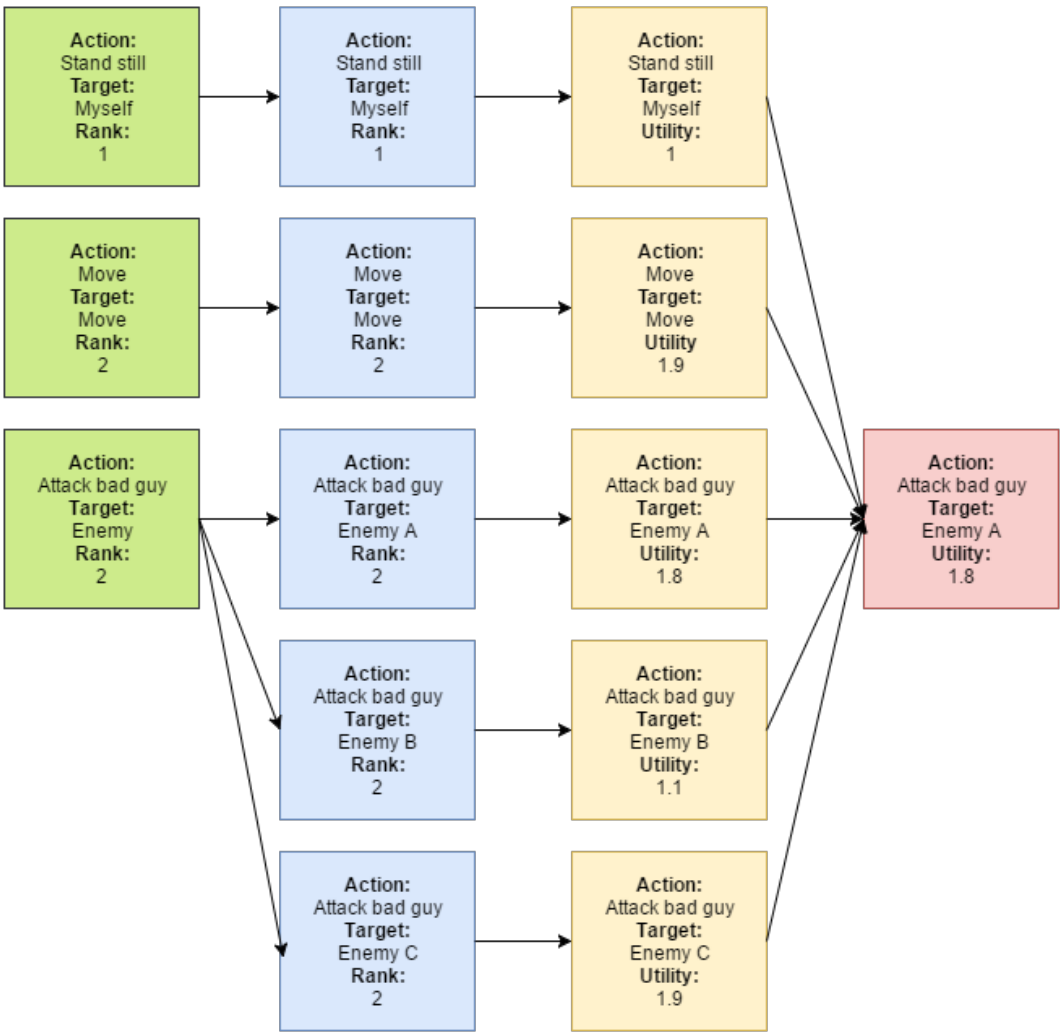


Figure 9: The picture depicts how a simple example of how a final decision is generated. The **Green** boxes represent all actions available in a specific utility frame. **Blue** boxes represent the same actions as the **Green** boxes but with a unique target received from a request to the Sensor component. **Yellow** boxes represent each **Blue** action evaluated in the utility frame containing each Action's final utility score. The **Red** boxes represent the final decision made by the utility frame from all the **Yellow** boxes by the weighted random method.

Chapter 7

Evaluation

7.1 Method

The Evaluation method for bots human-like appearance was heavily inspired by the Turing test for bots[2, 31]. The test was constructed with it as the base but was modified to answer the question: *Is the bots inseparable from its human counterpart by using the developed AI framework prototype?* This since Level Eight's primary use for the bots will be to substitute human players that have dropped out during a game.

Since the game has not yet been released, all test subjects are first time players and therefore have a high improvement curve during the test.

7.2 The Bot Test

The Bot test was developed for Level Eights game PROTOSTRIKE to meet their needs when it comes to the game AI of a bot. The main concern of the test is whether a bot of PROTOSTRIKE has the appearance of being a human and what characteristics might act in its favour and vice versa.

7.2.1 Setup

The number of players during each test is five: 2 human players each with a different finalised weapon (assault rifle and shotgun) and two bots also they with one assault rifle and one with the shotgun. The fifth player is the test subject, and that person can make a choice freely between weapons every game round. All the players except the test subject get randomly one of the following names: **Mr. Blue**, **Mr. Red**, **Mr. Brown**, and **Mr. Pink**. All human players (test subjects and test players) are instructed to play the game to win with all functionality in the game at their disposal.

Firstly each test subject has a chance to try the game with only stationary opponents with the support of an experienced player. This is done so that they learn all the basic actions of the game.

7.2.2 Test format

The test is performed by letting a test subject play the game five rounds of a standard FFA match using the map *Smugglerbase* with five players. The following format is used during the test:

- Each round is between the two human players, the two bots, and the test subject.
- The test subject may only interact with the other players by using the inputs available in the game PROTOSTRIKE.
- After a round is finished the test subject shall fill out an evaluation with haste.
- Each Player task during a game round is to try to win the game including the test subject.
- The test subject shall have no incentive to do good or bad at the test.
- The underlying representation supplied to humans and bots may differ.
- The test subject shall be aware of all aspects of the test.

The evaluation after each round of the other players' human-like appearance is given by the question *How human-like is the player (Player name)*? For each player with the following scale [31]:

1. This player is a not very human-like bot.
2. This player is a fairly human-like bot.
3. This player is a quite human-like bot.
4. This player is a very human-like bot
5. This player is human.

With the follow-up question *What characteristics of this player made you pick this answer?*, the test subjects can give an explanation to their answers in their words. See Appendix B to see the form for round one of the test, the question form for the other rounds looks the same.

Chapter 8

Results

Due to unfortunate events during the test phase, only six tests were conducted fully (with five rounds in each test), and one of the fixed human players had to be repeatedly switched during them. This has affected the results so that no significant statistical conclusions could be done. However, the comments from the tests performed have given keen insight into the bots strengths and weaknesses. The comments do also show some hints about what a player might judge as human behaviour and what they judge as bot behaviour even when their conclusions sometimes were wrong.

During the tests, **MR. Blue** was a bot with the assault rifle, **MR. Red** was a bot with the shotgun, **MR. Pink** was a human with the assault rifle and **MR. Brown** was a variety of humans from Level Eight which all used a shotgun. The results are presented in detail in Appendix C.

8.1 Ratings

When comparing the average scores of the final rounds of all tests, the bots have equal or fewer ratings than their human counterparts. Why the comparison is done only for the final round and not for all rounds are mainly because of the comments given during the earlier rounds, stating that the test subjects were unsure of their surroundings as most of them where new players of the game. Mainly, the test subjects were not able to focus on the other players during the first rounds to make decisive decisions revolving the other players.

Even though no bot on average acted more human-like than a real human player. On many occasions a bot and a human got the same score on average and on some occasions, test subjects thought that the bots were human and the human players were bots.

The scores and the average for each test are listed at the end of Appendix C.

8.2 Comments

8.2.1 Comments about the bots

Some of the test subjects were fooled by the bots but most were not. The following comments were given by the test subjects which rated the bots accurately:

- ”Focus on one target until the die did not notis me at all”

- ”Many kills but but did not followed me”
- ”I shot at him and did not run or try to shot at me back”
- ”Shot me but never hard target me easy to run out”
- ”It got stuck in a spot and didn’t really attempt to get away.”
- ”He got stuck in the enviroment a bit much”

The two main tells for a bot seems to be their movement by getting stuck on the environment and how they handle focus on other players regarding when to follow and when to flee.

The following are some of the comments from the test subjects that guessed that a bot was a human player:

- ”He felt varied”
- ”Lagom klumpig för att inte vara en dator” (Moderately clumsy enough not to be a computer)
- ”Hur spelaren rör sig på kartan” (How player moved on the map)
- ”Undvek och arbetade runt hinder på eget sätt” (Avoided and worked around the obstacles in his own way)

Apparently acting with variety and not always functioning correctly with good movement patterns are a way to act human-like according to some of the test subjects.

8.2.2 Comments about the human players

The following are some of the comments from test subjects that accurately guessed that a player was human:

- ”Many points and followed”
- ”Came after me”
- ”Felt too vindictive to be a cold emotionless robot. Killed me a lot. F u dude.”
- ”Follow me very hard to run out, but do not follow blind after me”
- ”Many kills”

The act of keeping focus in an intelligent way on other players seems to be mainly seen as a human trait in the tests.

The following are some of the comments from the test subjects that guessed that a human was a bot:

- ”Rör sig för bra på banan” (Moves too well on map)
- ”följer andra spelare för bra” (follows other players to well)

- ”Easy to run out and only focus on one player”
- ”There where two enemy and both did not focus me at all it was free dmg at them”

Being too good at the game seems to have made some of the test subjects think that some humans were in reality bots. That and being ignored seems to make a human look less human-like.

Chapter 9

Conclusions

9.1 Test results

9.1.1 The bots

During the tests the bot **MR. Red** using the shotgun was guessed to be more human than the other bot **MR. Blue** that was using the assault rifle. The only difference between their behaviours is that when they are fighting a player, a bot with a shotgun will prefer to stand closer to the enemy than a bot with an assault rifle. Moreover, many of the comments that were given when a bot got a low score mention that when they ran away, the bot loses interest and instead focus on other players. To conclude, MR. Red is perceived as more human-like most likely since MR. Red try to position itself closer to the player it fights, and therefore do not lose interest in the other player as quickly as MR. Blue.

Getting stuck on the environment is according to the test subjects another giveaway for a player being a bot. That was said even when human players got stuck as they apparently are guessed to be bots.

9.1.2 Bot characteristics

The characteristics that seem to be still lacking according to the test subjects are how the bots choose what target to follow and their decision between when to attack and when to flee.

The bots main positive aspect in being human-like seems to be that they do not feel static in their choices. This is mainly seen in comments were the test subjects got fooled by the bots. The reason that they do not behave in a static way when they make choices is that of the Utility AI making them act a bit randomly but always choosing a correct behaviour to do.

9.1.3 Results

The test results were according to Level Eight a success and the tests show that some of the test subjects were sometimes fooled which was part of the goal for Level Eight.

Given time and more testing, behaviours of the bots could be added and optimised to increasingly fool human players until a player cannot tell the difference.

9.2 Problems

9.2.1 Game AI information

Due to the Game development industry does not write articles of scientific nature many of the results found for techniques are bias in the way they are written as they are merely personal experiences from often one person. Because of this, many times the presented information about a technique does not focus on performance issues but rather on how easy it is to implement with other systems in a game. As ease of implementation seems to have more importance in the industry, this thesis has been tilted towards that mindset.

That being said more papers and articles with a larger focus on performance measurements and not usability would be great to the industry as a whole.

9.2.2 Time estimates of the thesis

This thesis started when Level Eight already had begun development on PROTOSTRIKE they already had the other AI system partly developed for the bots. In many ways, this has simplified the process as many behaviour implementations already existed, but this lead to some misassumption on how much time implementation would take.

Some of these assumptions later turn out to be false and this, in turn, made the implementation time to take longer than earlier planned.

9.2.3 Behaviours

Both in an academic sense and a practical sense building and understanding which behaviours should be implemented in the bots was and still is not an easy task to do. Many times trial and error, and also watching how humans play the game is the only way to realise how a behaviour should be implemented and when it should be used. This way of gaining knowledge in how behaviours should work is quite time-consuming and also really hard to measure when a behaviour is finished.

9.3 Future Work

9.3.1 Performance

The AI framework prototype has many improvements to be made. Many of them revolving performance improvements since the code was written to prove the concepts of the thesis and not only for performance.

9.3.2 Configuration

The Actions and Axis were all handwritten in *JSON*. One future feature should be to make a tool so that it is easy and quick to tweak and alter the behaviour of a bot. The tool should include the following:

- Drag and drop features.
- Spell checking.
- Curve visualisation for Axis in Axis Packets.

- Verification checks that an Action's corresponding Axis exist.
- Auto-complete when adding Axis to an Action.

A tool with the features listed above would reduce the time it would take to create new bot types significantly.

9.3.3 Balancing

Due to the game still being in development, weapons and character attributes are still being modified which affects the game-play in different amounts. This, in turn, means that the bots will have to be modified in the same pace as the rest of the game.

9.3.4 Scaling difficulty

A future improvement could be to implement ways to scale a bots difficulty levels by changing how often it is allowed to do new behaviours and also what configuration of behaviours it has available. This would make that new players meet bots that are not so difficult to beat and therefore to make the game more enjoyable.

9.3.5 Lack in behaviour

As seen in the tests, focus on hunting or fleeing has been one major flaw in the bots human-like nature. This needs to be corrected in the future if Level Eight wants the players to be genuinely fooled by the bot.

References

[1] Level eight — about. http://www.leveleight.se/about/. (Accessed on 01/14/2017).

[2] Philip Hingston. A turing test for computer game bots. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(3):169–186, 2009.

[3] Gregory and Jason. *Game engine architecture*. CRC Press, 2009. ISBN 1568814135.

[4] John Haas. *A History of the Unity Game Engine*. PhD thesis, Worcester Polytechnic Institute, 100 Institute Rd, Worcester, MA 01609, USA, 2014. (Accessed on 09/07/2016).

[5] Unity3d. https://unity3d.com. (Accessed on 09/07/2016).

[6] Why use C# for mobile game development with unity3d? http://www.biggshark.com/why-using-c-with-unity-is-better-than-boo-and-js-for-your-next-mobile-game (Accessed on 09/13/2016).

[7] Microsoft Corp.introduction to the C# language and the .NET framework. https://msdn.microsoft.com/en-us/library/z1zx9t92.aspx. (Accessed on 09/07/2016).

[8] Claude E. Shannon. Xxii. programming a computer for playing chess. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 41(314):256–275, 1950.

[9] Jakob Rasmussen. Jakob Rasmussen's Blog - Are Behavior Trees a Thing of the Past? Gamasutra, April 2016. http://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php. (Accessed on 09/06/2016).

[10] Ian Millington and John Funge. *Artificial intelligence for games*. CRC Press, 2016. ISBN 0123747317.

[11] Harry Henderson. *Encyclopedia of computer science and technology*. Infobase Publishing, 2009. ISBN 0816063826.

[12] Michael Dawe, Steve Gargolinski, Luke Dicken, Troy Humphreys, and Dave Mark. Behavior selection algorithms: An overview. In Steve Rabin, editor, *Game AI Pro: Collected Wisdom of Game AI Professionals*, chapter 4. CRC Press, 2013. ISBN 9781466565968.

[13] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[14] Damian Isla. Handling Complexity in the Halo 2 AI. In *Game Developers Conference (GDC)*, volume 2005, March 2005. http://www.gamasutra.com/view/feature/130663/gdc_2005_proceeding_handling_.php(Accessed on 09/02/2016).

[15] Jeff Orkin. Three states and a plan: The AI of FEAR. In *Game Developers Conference (GDC)*, volume 2006, 2006.

[16] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[17] Jochen Fromm. Types and forms of emergence. *arXiv: nlin/0506028*, 2005.

[18] Chris Conway, Peter Higley, and Eric Jacopin. Goal-Oriented Action Planning: Ten Years Old and No Fear! In *Game Developers Conference (GDC)*, volume 2015, March 2015. http://www.gdcvault.com/play/1022020/Goal-Oriented-Action-Planning-Ten(Accessed on 09/02/2016).

[19] Daniel Read. Utility theory from Jeremy Bentham to Daniel Kahneman. 2004.

[20] David Graham. An introduction to utility theory. In Steve Rabin, editor, *Game AI Pro: Collected Wisdom of Game AI Professionals*, chapter 9. CRC Press, 2013. ISBN 9781466565968.

[21] Kevin Dill and Dave Mark. Embracing the Dark Art of Mathematical Modeling in AI. In *Game Developers Conference (GDC)*, volume 2012, March 2012. http://gdcvault.com/play/1015683/Embracing-the-Dark-Art-of(Accessed on 09/02/2016).

[22] Dave Mark and Mike Lewis. Building a Better Centaur: AI at Massive Scale. In *Game Developers Conference (GDC)*, volume 2015, March 2015. http://www.gdcvault.com/play/1021848/Building-a-Better-Centaur-AI(Accessed on 09/02/2016).

[23] Gilbert Laporte. A short history of the traveling salesman problem. *Canada Research Chair in Distribution Management, Centre for Research on Transportation (CRT) and GERAD HEC Montréal, Canada*, 2006.

[24] Nathan R. Sturtevant. Choosing a search space representation. In Steve Rabin, editor, *Game AI Pro: Collected Wisdom of Game AI Professionals*, chapter 18. CRC Press, 2013. ISBN 9781466565968.

[25] Phil Carlisle. A simple and robust knowledge representation system. In Steve Rabin, editor, *Game AI Pro: Collected Wisdom of Game AI Professionals*, chapter 34. CRC Press, 2013. ISBN 9781466565968.

[26] Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[27] Graham Oppy and David Dowe. The turing test. 2016. http://plato.stanford.edu/archives/spr2016/entries/turing-test/(Accessed on 10/05/2016).

[28] Fritz Heider and Marianne Simmel. *An experimental study of apparent behavior*, volume 57. JSTOR, 1944. See also:https://www.youtube.com/watch?v=n9TWwG4SFWQ.

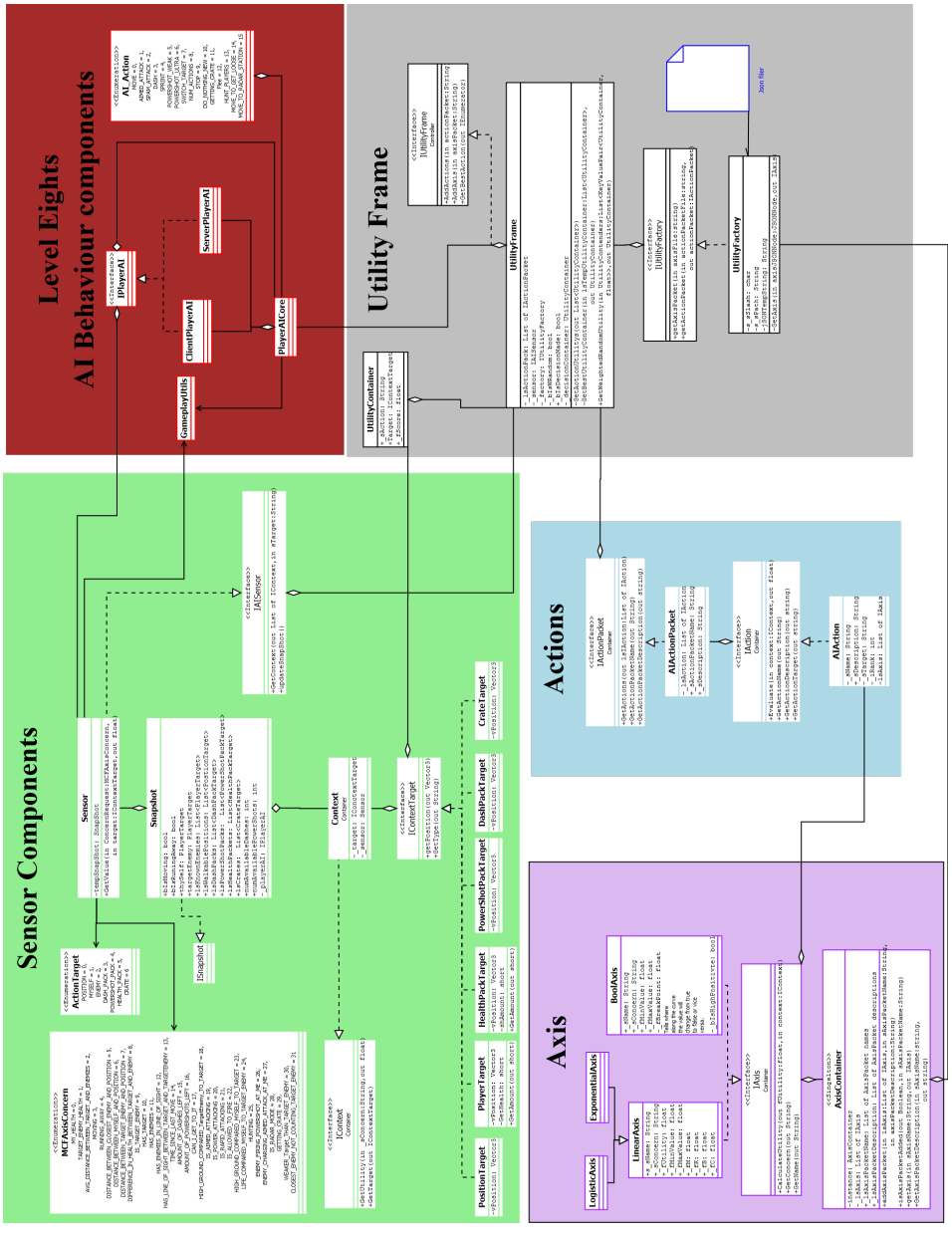
[29] Jamey Pittman. The pac-man dossier, Gamasutra. http://www.gamasutra.com/view/feature/3938/the_pacman_dossier.php?print=1. (Accessed on 09/19/2016).

[30] David Mark. 2016. Personal communication.

[31] Philip Hingston. A new design for a turing test for bots. In *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*, pages 345–350. IEEE, 2010.

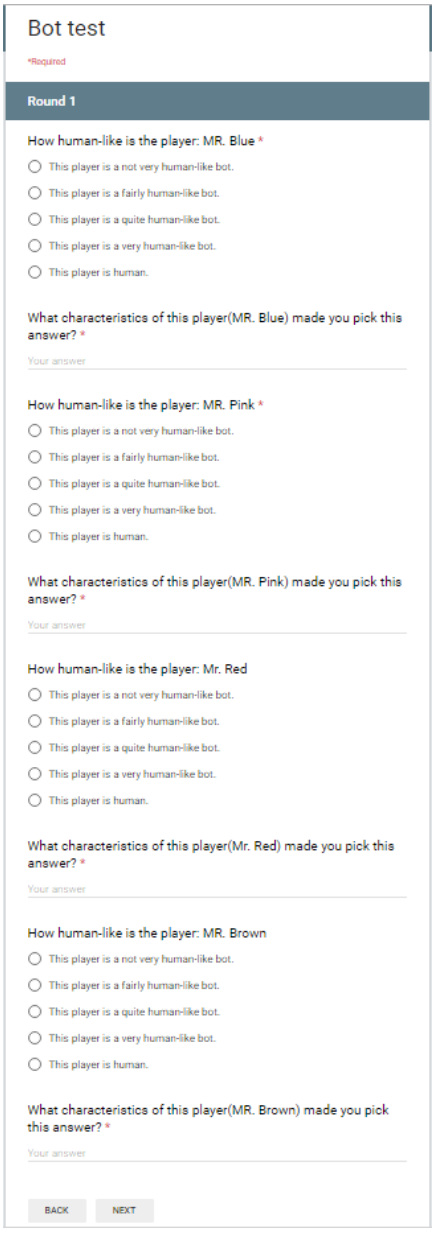
Appendix A

UML-diagram of the AI prototype framework



Appendix B

Player evaluation form round 1 of the bot test



Bot test

*Required

Round 1

How human-like is the player: MR. Blue *

- This player is a not very human-like bot.
- This player is a fairly human-like bot.
- This player is a quite human-like bot.
- This player is a very human-like bot.
- This player is human.

What characteristics of this player(MR. Blue) made you pick this answer? *

Your answer

How human-like is the player: MR. Pink *

- This player is a not very human-like bot.
- This player is a fairly human-like bot.
- This player is a quite human-like bot.
- This player is a very human-like bot.
- This player is human.

What characteristics of this player(MR. Pink) made you pick this answer? *

Your answer

How human-like is the player: Mr. Red

- This player is a not very human-like bot.
- This player is a fairly human-like bot.
- This player is a quite human-like bot.
- This player is a very human-like bot.
- This player is human.

What characteristics of this player(Mr. Red) made you pick this answer? *

Your answer

How human-like is the player: MR. Brown

- This player is a not very human-like bot.
- This player is a fairly human-like bot.
- This player is a quite human-like bot.
- This player is a very human-like bot.
- This player is human.

What characteristics of this player(MR. Brown) made you pick this answer? *

Your answer

BACK NEXT

Appendix C

Bot test form results

A note to the tests: **MR. Blue** and **MR. Red** are bots and **MR. Pink** and **MR. Brown** are human players.

C.1 Round 1

Test subject	How human-like is the player: MR. Blue	What characteristics of this player(MR. Blue) made you pick this answer?
1	This player is a quite human-like bot.	I did not met him verry often and cant make a good jugement
2	This player is a not very human-like bot.	För centrerad på spelaren
3	This player is a quite human-like bot.	I did not motis there names bur two of them where kind of human-like but two act like a bot
4	This player is a quite human-like bot.	I felt targeted
5	This player is a quite human-like bot.	I did not interact with mr blue
6	This player is a quite human-like bot.	Could not remember the names of the players who shot me

Test subject	How human-like is the player: Mr. Red	What characteristics of this player(Mr. Red) made you pick this answer?
1	This player is a not very human-like bot.	He got stuck in the enviroment a bit much
2	This player is human.	Undvek och arbetade runt hinder på eget sätt
3	This player is a fairly human-like bot.	I did not motis there names bur two of them where kind of human-like but two act like a bot
4	This player is a not very human-like bot.	I don't remember seeing mr red at all
5	This player is a quite human-like bot.	I did do not remember facing mister red
6	This player is a quite human-like bot.	Same as above

Test subject	How human-like is the player: MR. Pink	What characteristics of this player(MR. Pink) made you pick this answer?
1	This player is human.	His score is way higher than the reste and he Dodge a bit better
2	This player is a fairly human-like bot.	Också väl centrerad på spelaren
3	This player is a quite human-like bot.	I did not motis there names bur two of them where kind of human-like but two act like a bot
4	This player is a fairly human-like bot.	Same, i felt targeted
5	This player is human.	He chased me
6	This player is a quite human-like bot.	Same as above

Test subject	How human-like is the player: MR. Brown	What characteristics of this player(MR. Brown) made you pick this answer?
1	This player is a quite human-like bot.	Same as blue
2	This player is a not very human-like bot.	Se blå
3	This player is a fairly human-like bot.	I did not motis there names bur two of them where kind of human-like but two act like a bot
4	This player is a quite human-like bot.	Like with the orters, i felt targeted.
5	This player is a quite human-like bot.	Did not chase me even though I went low on ho,
6	This player is a quite human-like bot.	Same as above

C.2 Round 2

Test subject	How human-like is the player: MR. Blue	What characteristics of this player(MR. Blue) made you pick this answer?
1	This player is a quite human-like bot.	Did not met him much
2	This player is a not very human-like bot.	Såg inte blå på hela matchen?
3	This player is a fairly human-like bot.	Do not follow à easy taget like me
4	This player is a not very human-like bot.	Didn't get a chance to really notice him
5	This player is a quite human-like bot.	Did not interact with blue
6	This player is a not very human-like bot.	Did not had so many kills

Test subject	How human-like is the player: Mr. Red	What characteristics of this player(Mr. Red) made you pick this answer?
1	This player is a fairly human-like bot.	He tock a fight he was loosing
2	This player is a not very human-like bot.	För ta vapenhantering, bytte mål för snabbt
3	This player is a quite human-like bot.	Find a target and try to kill me until i run away
4	This player is a quite human-like bot.	Seemed to have a purpose.
5	This player is a quite human-like bot.	He seemed very consistent
6	This player is a not very human-like bot.	Did not had so many kills

Test subject	How human-like is the player: MR. Pink	What characteristics of this player(MR. Pink) made you pick this answer?
1	This player is a very human-like bot.	He evaded and such quite well
2	This player is a quite human-like bot.	Rörelsemönster
3	This player is a fairly human-like bot.	Act better that pink but still do not act like à human
4	This player is a quite human-like bot.	Movement seemed quite human
5	This player is a quite human-like bot.	Did not interact with pink
6	This player is human.	War top tre

Test subject	How human-like is the player: MR. Brown	What characteristics of this player(MR. Brown) made you pick this answer?
1	This player is a very human-like bot.	Same as brown
2	This player is a quite human-like bot.	Rörelsemönster och hur karaktären använde väggar som skydd
3	This player is human.	When he found me he is on me and do not let me go befor i die
4	This player is a quite human-like bot.	Movement seemed quite
5	This player is human.	He was really bad when I first encountered him and then he played a lot better
6	This player is human.	Was frist place

C.3 Round 3

Test subject	How human-like is the player: MR. Blue	What characteristics of this player(MR. Blue) made you pick this answer?
1	This player is human.	Cant tell
2	This player is a not very human-like bot.	Sågs inte av igen?
3	This player is a not very human-like bot.	I shot at him and did not run or try to shot at me back
4	This player is a quite human-like bot.	Eh. Seemed real enough
5	This player is a quite human-like bot.	We did not interact
6	This player is a quite human-like bot.	Came after to shot me

Test subject	How human-like is the player: Mr. Red	What characteristics of this player(Mr. Red) made you pick this answer?
1	This player is human.	Cant tell
2	This player is human.	Rörelsemönster, kan inte sätta fingret på det
3	This player is a fairly human-like bot.	Shot me but never hard target me easy to run out
4	This player is a not very human-like bot.	It got stuck in a spot and didn't really attempt to get away.
5	This player is a quite human-like bot.	I do not remember interacting with red
6	This player is a quite human-like bot.	Not sure have not really met the player

Test subject	How human-like is the player: MR. Pink	What characteristics of this player(MR. Pink) made you pick this answer?
1	This player is human.	Cant tell
2	This player is a not very human-like bot.	Fastnade på ett område när inga fiender va nära nog
3	This player is human.	Follow me very hard to run out, but do not follow blind after me
4	This player is a quite human-like bot.	Quite real-seeming.
5	This player is a quite human-like bot.	We did not I teract
6	This player is human.	Many kills

Test subject	How human-like is the player: MR. Brown	What characteristics of this player(MR. Brown) made you pick this answer?
1	This player is human.	Cant tell
2	This player is a fairly human-like bot.	Vapenhanteringen kändes inte helt mänsklig
3	This player is a fairly human-like bot.	Easy to run out and only focus on one player
4	This player is a quite human-like bot.	No complaints
5	This player is human.	He seemed good and intelligent
6	This player is human.	Most kills

C.4 Round 4

Test subject	How human-like is the player: MR. Blue	What characteristics of this player(MR. Blue) made you pick this answer?
1	This player is human.	No obvius ”bot-like” behaviors
2	This player is human.	Hur spelaren rör sig på kartan
3	This player is a fairly human-like bot.	There where two enemy and both did not focus me at all it was free dmg at them
4	This player is a quite human-like bot.	No hints either way really
5	This player is a quite human-like bot.	I do not remember interacting
6	This player is a quite human-like bot.	Have not met the player

Test subject	How human-like is the player: Mr. Red	What characteristics of this player(Mr. Red) made you pick this answer?
1	This player is human.	Same
2	This player is human.	Lagom klumpig för att inte vara en dator
3	This player is a quite human-like bot.	Change party targets when i start to hit him
4	This player is a quite human-like bot.	No hints
5	This player is a fairly human-like bot.	He did seem to have a predictive move,net pattern
6	This player is a fairly human-like bot.	Many kills but but did not followed me

Test subject	How human-like is the player: MR. Pink	What characteristics of this player(MR. Pink) made you pick this answer?
1	This player is human.	Same
2	This player is a not very human-like bot.	följer andra spelare för bra
3	This player is a fairly human-like bot.	There where two enemy and both did not focus me at all it was free dmg at them
4	This player is a quite human-like bot.	No hints
5	This player is human.	He see,Ed to position well and adapt to the situation
6	This player is human.	Came after

Test subject	How human-like is the player: MR. Brown	What characteristics of this player(MR. Brown) made you pick this answer?
1	This player is human.	Same
2	This player is a not very human-like bot.	För snabb på att undvika attacker, spelar med handikapp av något slag?
3	This player is a very human-like bot.	Change targets and had most humen but when i was to close to him he was kind like à bot
4	This player is human.	Felt too vindictive to be a cold emotionless robot. Killed me a lot. F u dude.
5	This player is a quite human-like bot.	Ido not remember interacting with mr brown
6	This player is human.	Came after me

C.5 Round 5

Test subject	How human-like is the player: MR. Blue	What characteristics of this player(MR. Blue) made you pick this answer?
1	This player is human.	To fast for me to notice anything that i see as obvious stupid
2	This player is a quite human-like bot.	byter mål för ofta
3	This player is a fairly human-like bot.	Focus on one target until the die did not notis me at all
4	This player is a fairly human-like bot.	2 efficient 2 b real
5	This player is a quite human-like bot.	I do not remember much of mr blue
6	This player is a quite human-like bot.	Did not follow

Test subject	How human-like is the player: Mr. Red	What characteristics of this player(Mr. Red) made you pick this answer?
1	This player is human.	Same
2	This player is a very human-like bot.	..
3	This player is a not very human-like bot.	This player sucks , kind of random patern
4	This player is a quite human-like bot.	No indications either way
5	This player is human.	He felt varied
6	This player is a quite human-like bot.	Did not follow

Test subject	How human-like is the player: MR. Pink	What characteristics of this player(MR. Pink) made you pick this answer?
1	This player is human.	Same
2	This player is human.	Klumpig i närstrider
3	This player is a fairly human-like bot.	Focus on one target ,did not notis me at all
4	This player is a quite human-like bot.	No indications either way
5	This player is a quite human-like bot.	I do not remember
6	This player is human.	Many points and followed

Test subject	How human-like is the player: MR. Brown	What characteristics of this player(MR. Brown) made you pick this answer?
1	This player is human.	Same
2	This player is a not very human-like bot.	Rör sig för bra på banan
3	This player is a very human-like bot.	Play smart and chnage targets , have kind à random patern
4	This player is a quite human-like bot.	No indications either way
5	This player is a quite human-like bot.	I do not remember him
6	This player is human.	Many points and followed

C.6 Bot test score summary

C.6.1 Scores for round 1

Test subject	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	3	5	1	3
2	1	2	5	1
3	3	3	2	2
4	3	2	1	3
5	3	5	3	3
6	3	3	3	3

C.6.2 Scores for round 2

Test subject	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	3	4	2	4
2	1	3	1	3
3	2	2	3	5
4	1	3	3	3
5	3	3	3	5
6	1	5	1	5

C.6.3 Scores for round 3

Test subject	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	5	5	5	5
2	1	1	5	2
3	1	5	2	2
4	3	3	1	3
5	3	3	3	5
6	3	5	3	5

C.6.4 Scores for round 4

Test subject	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	5	5	5	5
2	5	1	5	1
3	2	2	3	4
4	3	3	3	5
5	3	5	2	3
6	3	5	2	5

C.6.5 Scores for round 5

Test subject	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	5	5	5	5
2	3	5	4	1
3	2	2	1	4
4	2	3	3	3
5	3	3	5	3
6	3	5	3	5

C.6.6 Average scores for the final round

Round	MR. Blue(bot)	MR. Pink(human)	Mr. Red(bot)	Brown(human)
1	2.6	3.3	2.5	2.5
2	1.8	3.3	2.2	4.2
3	2.7	3.7	3.2	3.7
4	3.5	3.5	3.3	3.8
5	3	3.8	3.5	3.5