PROJECT REPORT

Designing a simple video game with Artificial Intelligence

MSc Computer Science

Department of Computer Science and Information Systems

Birkbeck, University of London

Sylwester Stremlau

Sstrem01

Disclaimer

I have read and understood the sections of plagiarism in the School Handbook and confirm that the work is my own, with the work of others clearly acknowledged. I give my permission to submit my submission to the plagiarism testing database that the College is using and test it using plagiarism detection software, search engines or meta-searching software.

SYLWESTER STREMLAU

Abstract

Glossary

Artificial Intelligence (AI)

Artificial Intelligence is the computer simulation of human behaviour and intelligence. It includes various processes like learning, adapting and thinking to act like a real human. There are various types of AI, from weak (also known as narrow) designed to do a certain task, to more complex, like strong AI that acts and thinks more like a real human being. (Rouse, 2019)

Rule Based Artificial Intelligence

Rule based AI is a way to program AI to make decisions based on rules created and modified by a human expert.

Machine Learning (ML)

Machine Learning is an implementation of Artificial Intelligence to learn and train itself using data. Machine Learning usually needs training data fed and supervised by an engineer, to look for patterns and similarities so it can make more complex and informed decisions on new data. (Nagy, 2018)

Video game engine

A video game engine is a software environment that provides functionality to allow creating and developing games more easily. It provides stuff like rendering engine, physics engine, sound, scripting, animation and other essential functions needed to make a game. (Ward, 2019)

Game Loop/Gameplay

Game loop is a very basic loop that controls the flow of the game. A typical game loop is: process inputs, update game and render objects on the screen. (Bethke, 2003)

Game physics

Game physics are a simulation of physics that follow more or less how objects behave in real life. Games do not have to follow real life physics but they usually have their own rules that are consistent throughout the game. (Bethke, 2003)

Graphical User Interface (GUI)

Graphical user interface is a way for user to interact with an application program through graphical items and indicators. (Bethke, 2003)

CONTENTS

CHAPTER 1 - INTRODUCTION

State the problem you are trying to solve

Why is it worth tackling?

What approaches are available (briefly)?

What approaches have you chosen?

Any special knowledge you presume of the reader

Any special typography or terms

A “road map” of the report . . .

CHAPTER 2 – BACKGROUND

Any information the reader requires in terms of techniques/ technology that isn’t part of the programme you have studies.

Methodology

Research philosophy

CHAPTER 3 – ANALYSIS, REQUIREMENTS AND DESIGN

The game is a mix of various game types including shooters, fighting and sports games, with a top down camera view. It was heavily influenced by games like Rocket League, FIFA and Hotline Miami. On a basic level it is very similar to a classic football game in which there are two teams, one ball and two goals. Each team tries to kick the ball into the opponents' goal to score a point, the team with the highest points at the end of the match wins.

However, the rules of the game are a lot different than in a classic football game, first, the players are allowed to attack the opponents and even get points when they manage to kill him. Players get weapons like a sword, gun and a shield to do so. The teams are also much smaller and should first be limited to only one player on a team to make it of manageable size to finish, especially since a lot of focus in this project is on the Artificial Intelligence aspect. Having bigger team size would introduce new ways that the AI should behave and new situation to which it would have to adapt, this would complicate the process and may make the game impossible to finish in a short span of time.

The rules:

- No offside – there will be 'walls' placed around the play area which the ball will bounce off and which the players will not be able to move beyond
- Team gets two points per goal - only if the ball is all the way in the opponents goal and touches the back wall of the goal
- Players can use shield, swords and guns to defend their goals
- The position of the players resets to their default position after each goal and the ball goes back to the middle
- Players can also use the weapons to attack other players to drain their health
- Players start with 100HP, 100Stamina and 999 Ammo for each weapon
- Players can pick up boxes that spawn in one of the six fixed positions on the map
- One box spawns every 10 to 30 seconds
- Maximum of three boxes can be present at the same time
- Once the players health reaches zero, they are reset to their default position and their health is set back to 100 and the opponents team gets 1 points
- Players can only use one weapon at the same time

The view

The 'camera' is pointed from above like in games like Grand Theft Auto, Hotline Miami, Overcooked or Undertale. This point of view and 2D graphics have been chosen because it is much easier and faster to work with and even though it may seem dated, the gaming community and the market for these type of

games is still very big. For example, Undertale, released in September 2015 was one of the best-selling games on the digital game market Steam, with over 530 thousand copies sold by the end of 2015 (Galyonkin, 2019). Undertale uses top down view and retro, 2D graphics in its gameplay and it does not stop people from buying and enjoying it.

The aim will be to make the game look and feel very intuitive so anyone can start it and after just few minutes, be able to understand the main goal of the game and what he needs to achieve in order to success in it. An example of a very intuitive and straightforward game that was used as an inspiration for this project is the Super Mario Bros platform game published by Nintendo released in 1985. This game is a perfect example of a game that is easy to get into and very difficult to master. It has very basic controls – arrows to move, A to jump and B to sprint or shoot. The goal of the game is to collect coins, jump on platform and try not to die by falling into an empty pit or getting killed by enemies. This can seem very basic but the game manages to achieve a lot with this basic premise and is still an inspiration to many games.

Controls

PICTURE

The game controls will be based on a gamepad to create a very best experience since the player should be able to move in every direction, not only left, right, up and down that the keyboard (arrows) allows. (Although it will still be possible to use keyboard and mouse to control the game). A full 360 movement can only be achieved with analogue sticks on a modern gamepad. User will be able to rotate the player object using the right analogue stick, and use the weapons using the other buttons. Since the user will be using his left hand to move, it would also make sense to use the left hand to trigger sprint, this should be achieved by using the L1 button.

Some less experienced players may have difficulty with controlling the movement and rotation separately therefore an option to allow the player to move and rotate with one stick will be added. This should make the game easier to control and would leave their right hand for other tasks like using the weapons. More experienced players may find this type of controls not be flexible enough because it restricts their movement a lot therefore they would be able to switch between the two. A similar approach has been used in a game called Rocket League where players can either have their camera set to be behind-the-player at all times or to have it focused on the main point of the game which is the ball. The first type of control allows the players to be able to look around the game area and be aware of what is going on around them but it also makes it harder to control the games because they have to manually keep looking at the ball – there is a little arrow pointing at the ball to make it easier to locate it. The second type is much easier to controls because the players only have to focus on the movement and on trying to score a goal which is the main point of the game.

The play area

Before explaining the game loop and the goal of the game, it makes most sense to first explain the game area. The game area is a classic football pitch that is a rectangle with a line in the middle separating it into two; each side 'belongs' to each team. It 'belongs' to a team in a sense that the specific team will only spawn on its half of the pitch - this part is just like in the football. There are also goals on each side to which the players have to kick the ball into to score a goal. A team gets one point only when the ball hits the back of the goal, so when the ball is only halfway inside the goal, the team will not get the point. [PICTURE]

The ball

At the beginning of the match the ball spawns exactly in the middle of the pitch and whenever it gets hit into the goal, it resets there. This gives each team a fair position from which they can start the game. Player can push the ball with their characters, use swords to hit the ball the travel much faster or use guns to hit (push) it from a distance. The players can score points either by killing the opponent team – for each kill they get one point. Players also get points for kicking the ball into the goal as mentioned above.

Size of the team

There is only one player on each team but the game could be expanded to more players, even up to 10 football although that would put a heavy load on the computer. This size of the team has been chosen because of the complexity of Artificial Intelligence on which this project is focusing on. Bigger size of the teams would make the AI very complex because it would have to dynamically adapt to a lot more situations. For example, in a one-on-one game the AI only needs to follow the ball, block its own goal and try to score the goal. In a two-on-two game the AI would have to do all that plus it would also have to make sure it does not get in the way of its own team mates. More team mates would also mean that in a single player mode (which is when one actual human plays the game) new AI would have to be created which would be in the team with the human player. This means that the AI would have to behave and react to the players' behaviour differently since a real human is much less predictable. Therefore, focus as of now is on a one-to-one game.

HP, SP, SS and Ammo

Players have few things that they need to keep track of, Health Points (HP), Stamina Points (SP), Shield Size (SS) and Ammo. The Health Points work very similar as in other games, the player starts with 100HP (which is also the maximum they can have), if it goes below that, they can pick up boxes which replenish their Health Points by a certain number. The amount of HP that each box replenishes is a random

number between 10 and 50. When the player HP goes to zero, his position is reset to its default position that is bound to the game character, and one point is given to the opposite team.

Weapons

To kill the opponents, players have few weapons to use – a sword, a pistol or a machine gun. A sword drains 4 to 6 points of stamina and deals 10 to 15 damage. Stamina points also drains when a player sprints and just like with health points, player starts with 100 and once it goes to zero, player cannot sprint or use sword. Stamina replenishes around 1 point per frame (multiplied by the time.deltatime). Going back to the weapons, players also have guns and each gun has a different speed, ammo, reload time and strength. Pistol for example is much slower and has a smaller magazine size than the Machine Gun but it deals more damage and pushes the ball harder. Machine guns on the other hand have big magazine sizes, that is, they can shoot more bullets before they need to be reloaded, and they are faster and have a shorter reload time. Players can pick up ammo from the random boxes that spawn every 10 to 40 seconds around the map.

Shield

The last thing that players have at their disposal is the shield which is a half circle that spawns in front of the character for a few seconds which gets smaller and smaller until it disappears completely. The shield lasts for around 2 seconds and can be used as much as possible. It acts just like a wall that is, whenever a ball hits it, it gets bounced back so it is very useful when defending a goal. Its size allows the player to push the ball away and with little bit of practice can be used to score points. Shield can also be used to block bullets since player gets no damage when the bullet hits the shield. The downside to the shield is that the player cannot use a gun or a sword during the time that the shield is up.

TABLE FOR POINTS

TABLE FOR GUNS

Walls

The goals

Ball

Players

-actions etc.

https://docs.unity3d.com/ScriptReference/Rigidbody2D.html

Colliders can be used as triggers which mea

Should include appropriate formal design diagrams/notation as necessary

Artificial Intelligence Design

Designing the artificial intelligence for a game like this creates a number of problems and questions that need to be taken care of. First issue to address is to find out how the AI will behave in a 1vs1 game, most of this project will focus on that part because this will be a foundation for other types of AI in game. Second issue is how the AI will behave in a team in a 2+vs2+ game; how it will recognize who are its teammates and how it will take advantage of playing in a team. The third issue is similar, but this time its teammate will be the user, so it will be important to solve how the AI will react to an unpredictable player. This section will address these issues and present the designs for each type of AI.

1 vs 1

This game mode will create a foundation for other game modes because this will be the default behaviour of AI. That means it will not have to worry about getting in the way of its teammates and figuring out what its teammates are doing or their positions. Its only objective will be to score a goal. Before understanding how everything will work together, it is important to understand the basic behaviours that the AI will have at its disposal.

Movement

Point of focus

The first issue will be to find out what the AI should focus on – the general direction that the player should look in. The obvious answer would be to look at the ball at all times and move towards it, this may seem reasonable but it would disable the AI to use weapons to attack the opponent, especially in a game with more than one player on each team, because it first needs to aim its gun in a direction that it wants to shoot. So a method will need to be created to allow it to look in any direction.

This creates another issue – the player should not always move in the direction that it is looking at. Especially since users also have the ability to aim and move separately as mentioned in the Player Controls section. So a separate method for movement should be created.

Using weapons and the shield is very straightforward since these methods should already be created as mentioned in a previous section. A way to call that method will only be needed – this could be done with a static method that takes the player game object and runs, for example, a UseSword method inside it that will be stored in Controls component.

Helper methods should be created that calculate the distance from the player object to another object, finds the rotation to a wanted object or finds the opponents' position. These should make it easier to create more sophisticated behaviours.

Now that the movement, rotation and weapons can be controlled, more complex behaviours can be created. The first, and most likely the most important behaviour, is to score a goal. This behaviour will have to be broken down into separate steps. The default weapon to use to shoot a goal is the sword, and to use a sword the player has to be close to the object that it wants to affect with this sword. Therefore, the first step would be to get close to the ball, this can be easily achieved by finding out the balls directions and moving towards it. Once the player is close to the ball, it should not use a sword yet because it could be positioned in a way that if he uses it, the ball will move in a different direction than desired as seen in figure …. So the second step would be to position itself around the ball so that the ball faces the goal. The last step would be to actually use the sword.

As it can be noted, there are a lot of tasks and calculations that need to be done before the task is completed. This philosophy of breaking the task into smaller pieces inspired by the micro productivity as explained in…. will be used throughout this project. Breaking the tasks can make them a lot easier to achieve and makes them much more flexible. For example, when the AI has chosen to score a goal but the variables have changed before it is finished with it and it is now not the most effective task to do, it would be able to easily abandon it but keep the information gathered in the previous task and do something else without having to start at the first step.

https://www.microsoft.com/en-us/research/project/microtasks-and-microproductivity/?irgwc=1&OCID=AID2000142_aff_7806_1246483&tduid=(ir__6n2pevwuwgkfrlspkk0sohz30n2xjqpnf02owvev00)(7806)(1246483)(%28d2de1c3646d9ad0bace4b864b91163dc%29%2881561%29%28686431%29%28at106140_a107739_m12_p12460_cGB%29%28%29)(d2de1c3646d9ad0bace4b864b91163dc)&irclickid=_6n2pevwuwgkfrlspkk0sohz30n2xjqpnf02owvev00#!publications

Behaviour	How it will be achieved (steps)	Steps needed	Comments
Shoot the ball at the goal	Get close to the ball, position itself, use a sword	3	
Attack the opponent with a sword	Find the opponents position, rotate to look at the opponent, move towards the opponent, use a sword	4	
Attack the opponent with a gun	Find the opponents position, look at the opponent, use a gun. Additional steps may be needed if the player has no ammo or needs to reload – these should however be done automatically in the UseGun method – or if the player has to switch weapons.	3 to 6	

Defend the goal using shield	Find the balls position, look at it, use a shield.	3	The shield should not be used when the player is facings its' team goal since it can risk an own goal.
Defend the goal not using the shield	Find the position of the ball, move towards it, and position itself so the ball is between opponents' goal and the player. Player could additionally use a sword to kick the ball away.	3	
Get a pickup box	Check if there are any pickup boxes around, check if any of them are needed (for example if the player has 100HP there is no need to pick up a health box), check if it is close enough, move to its position	3	It is important to make sure that the pickup box is not too far, otherwise the player is risking that the opponents team will score a goal

PICTURE OF THE GOAL

How it chooses what to do

Break it down into 4 parts based on the position of the ball

Break it down even further into another four task based on the position of the players and the ball

Choose the best task

PICTURES

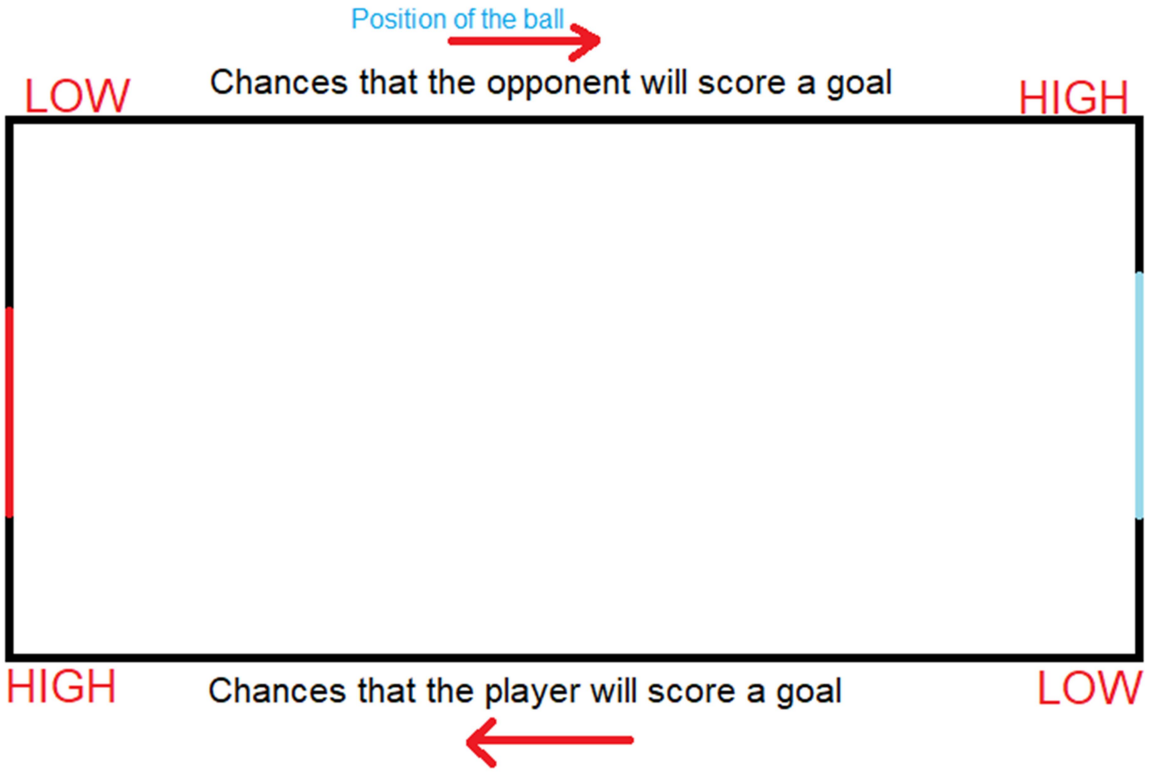
Each one of these behaviours creates a very basic behaviour tree that will be used in larger structures to create complex behaviour. The next step after creating the basic behaviours is to find a way in which the artificial intelligence will choose the most effective behaviour. This can also be done by using behaviour trees which are a way to structure the switching between various tasks easily. This means that each decision will have to be broken down into smaller parts.

The first step is to look at the behaviour more generally, as seen in fig … a basic behaviour is broken down into three or four smaller task, to keep things simple, a similar approach will be used. The general goals of the player are to:

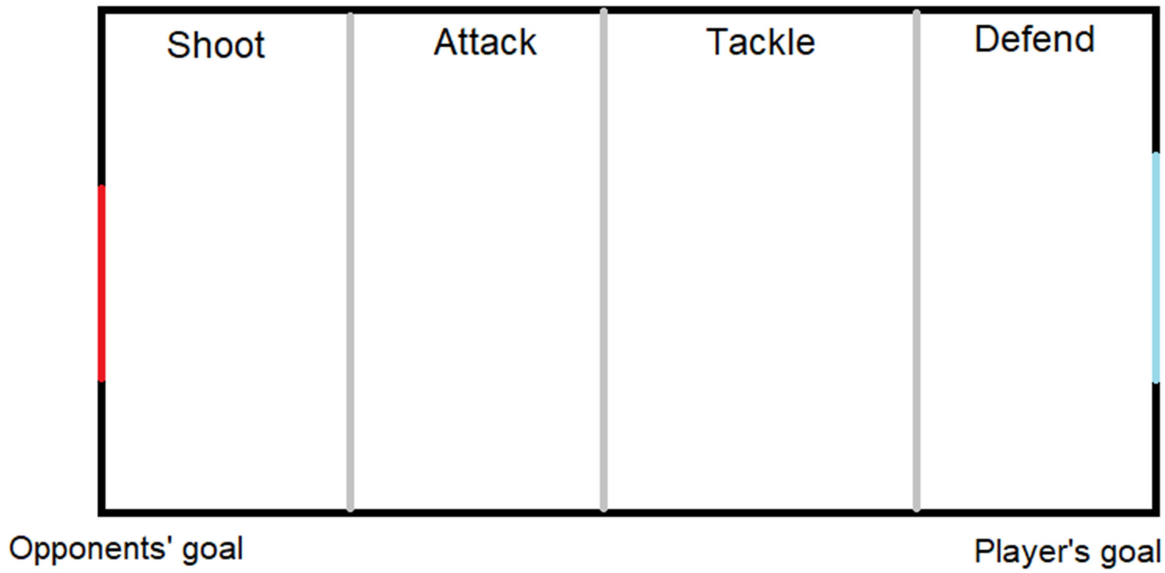
- Score a point
- Defend the goal

- Make it hard for the opponent to score a goal
- Do not die

The player cannot realistically try to all of those tasks at the same time so he may try to do them based on the position of the ball. For example, it should try to score a point when the ball is close to the opponents' goal. The chances that the opponent will score a goal decrease the further away the ball is from the player's goal and the chances that the player will score a goal increase the closer the ball is to the opponents' goal. All of these states and positions are relative to the player that the Artificial Intelligence is controlling and it is assumed that the player starts on the right side.



This means that the position of the ball can be used to help decide which one of the four tasks should be prioritised. So the play area has to be broken down into four states, these states are Shoot, Attack, Tackle and Defend. These will be the first step in the behaviour tree.



Shoot

In the shoot state the player should focus on scoring the goal by using any means necessary. The main point of focus would be to position itself in a position that will give him the most chances of scoring a goal. It should not focus getting pickup boxes unless its health is really low and he is really close to it.

Attack

In this state, the most important task will be to take the ball away from the player, attack the player to drain his health points and get a pick up box if needed. This state is more flexible than the shoot state because there are low chances that the opponent will score a goal.

Tackle

Similarly as to the attack state, player has much more freedom although it should be more careful because the chances that the opponent will score a goal are higher. In a game with more than one player on one team, one of the team mates should start to position itself and get ready to defend the goal while the other player should try to kick the ball away towards the opponents' goal.

Defend

This state along with the shoot state are the most crucial for the outcome of the game because they decide which of the players will score the most points. In this state the main task should be to defend

the goal and make it as hard as possible for the opponent to score a goal. Players' main task should be to use the shield to block the goal and sword to kick the ball away from the goal. Unless the player has really low health points, he should not try and get pick up boxes.

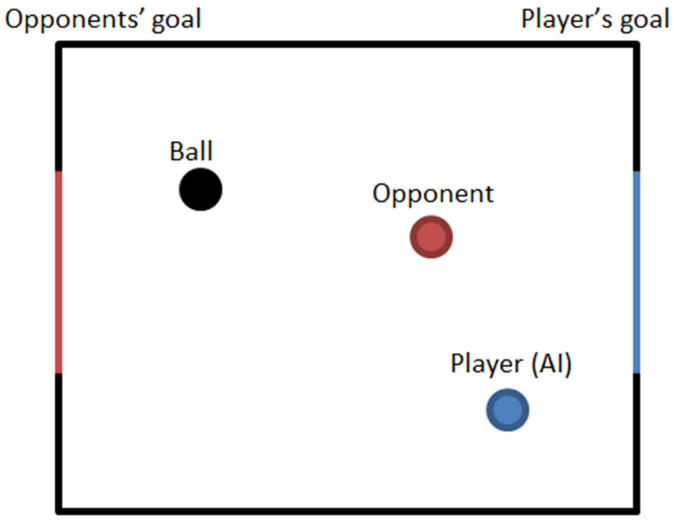
Next step would be to find a way how to break down the tasks even further. Here the position of the player and the opponents could come in helpful. Since it makes no real difference where the players and the ball are position on the y axis it will be ignored for now. It may make some difference in a game where there is more than one player in the team but for now the focus will be on a 1vs1 game.

Therefore the AI player should behave differently depending on its x position relative to the ball, opponent and the opponents' goal. For example, if the ball is in the Defend state, and the ball is between the opponent and the player, and the player's goal like in fig… Using a shield in that position may not be most effective because the player may accidentally push the ball in his own goal.

Or, if the ball is in the attack state and the player and the opponent are to the right of the ball and the ball is between opponents' goal and them. The player could quickly grab a pick up box without risking that the opponent will score a goal because the opponent has no way of scoring a goal from that position.

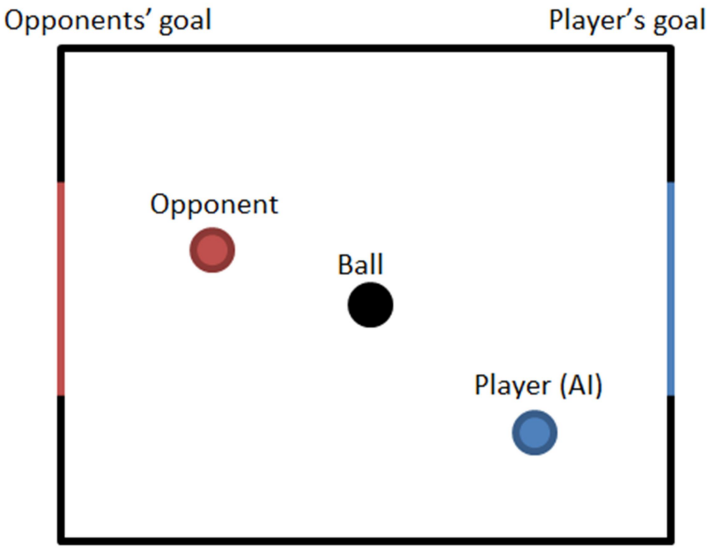
These positions are named Zero Position, One Position, Two Position and Three Position. Each of these states will prioritise a different behaviour just like the first four states did.

Position Zero



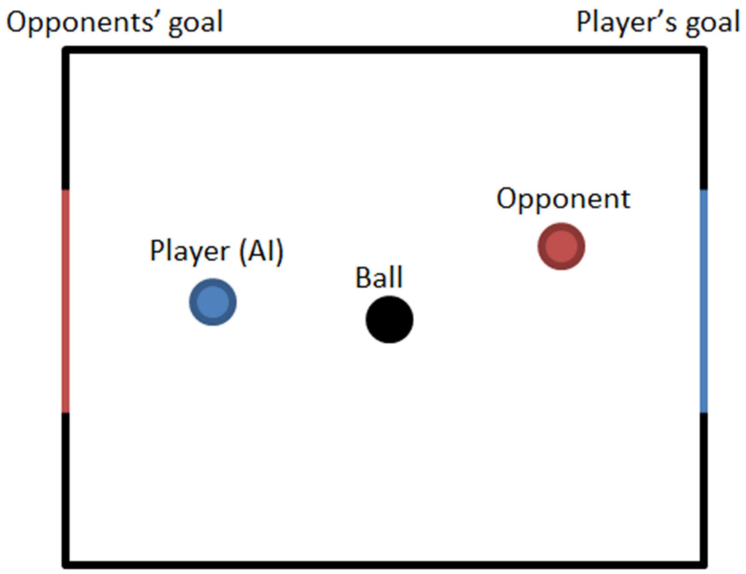
This position is advantageous for the player because there is no obstacle between the ball and the ball which makes it easy for him to score a goal. Depending on the state, the player in this position could focus on scoring a goal or, if the ball is in the defend position; he could be able to quickly grab a pick up box or attack the opponent with no risk.

Position One



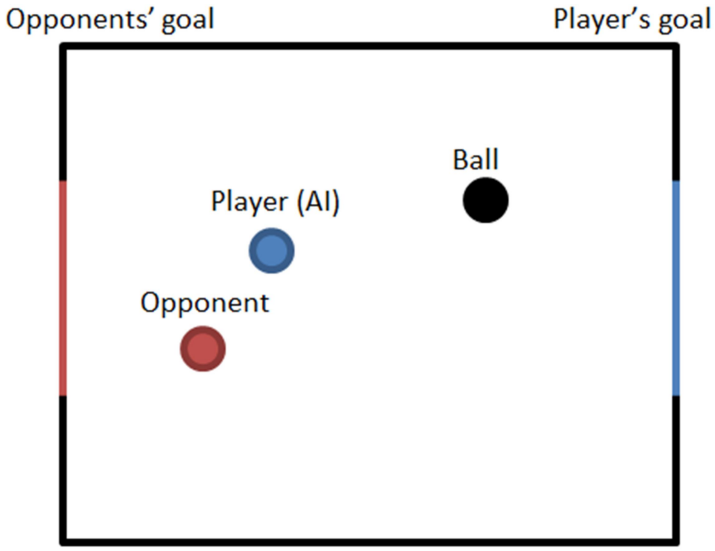
This position is neither good nor bad for the opponent and the player because the opponent and the player cannot score a goal. Usually, the most effective task to do would just be to try and turn around.

Position Two



This position is similar to the One Position but this time both the player and the opponent are at risk. This position is the most neutral and what the player does is heavily dependent on the state that the ball is at. For example, in a Defend State the player should try to use the shield to defend its own goal but in the Attack State it should try to score a goal.

Position Three

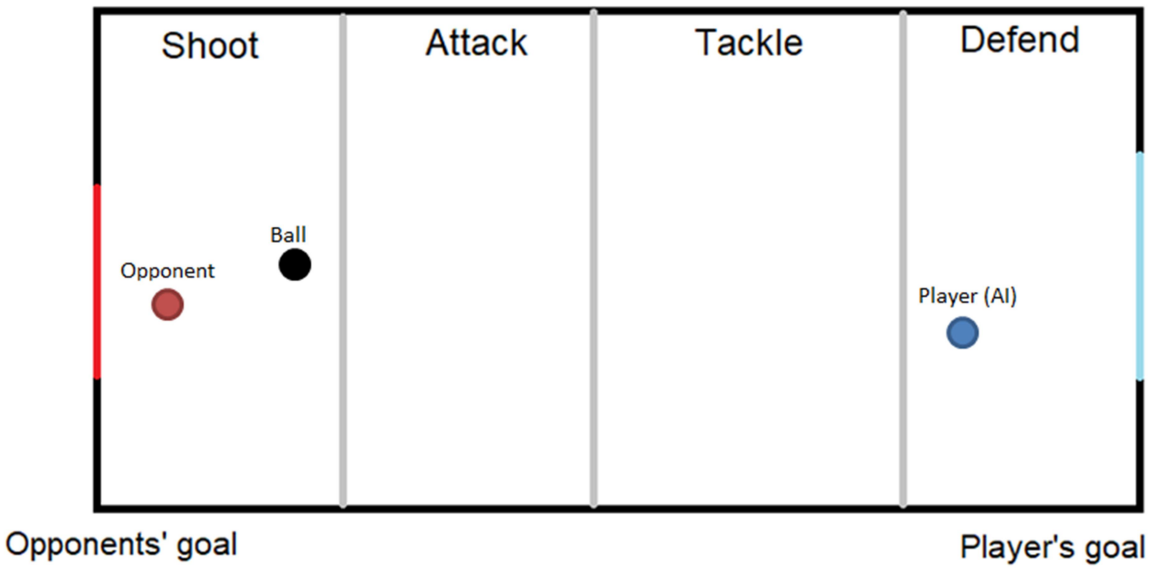


The last position is the most advantageous for the opponent because there are no obstacles between the player's goal and the ball so the opponent can easily kick it, even from a far, and score a goal. The

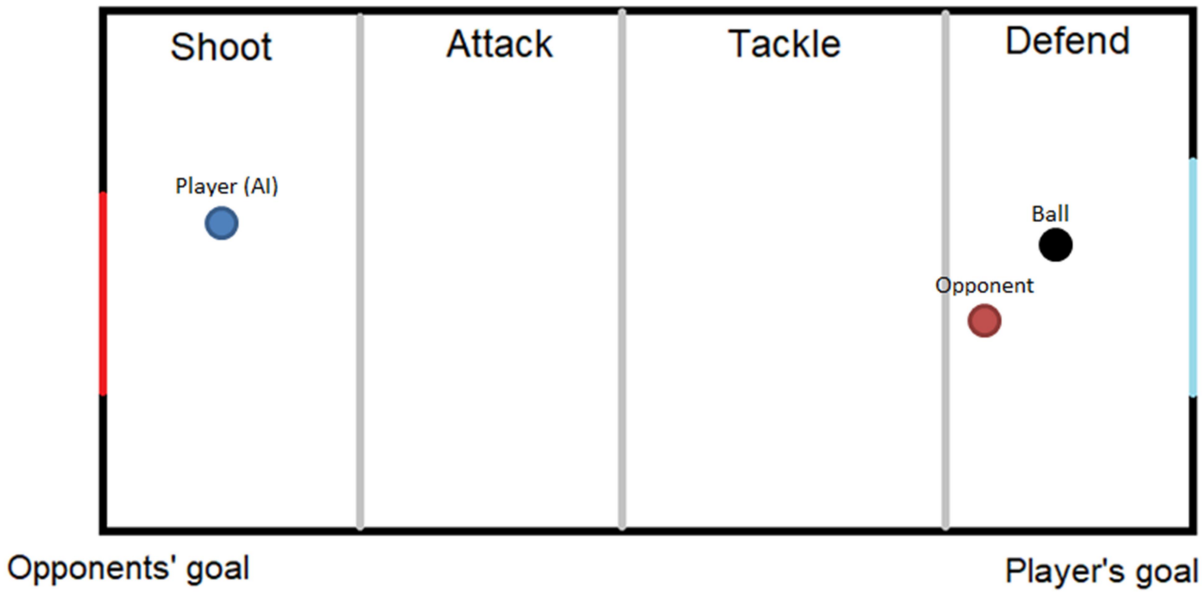
player in that position should almost always try to get close to the ball and get between it and its own goal, i.e. get into the Two Position.

https://arxiv.org/pdf/1709.00084.pdf

To cover almost every possible situation that the player may find itself in, there is one more issue to consider. That is the distance from the player to the ball; its behaviour should be slightly different if the player is far from ball. For example, if the ball is in the shoot area, and the player is in the defend area like shown in the figure … instead of just sprinting to the ball and risking that the opponent will kick the ball away, the player may use a gun to try to shoot the ball inside the goal from a distance. PICTURE



Similarly, if the ball is in the defend area, the player is in the shoot area and the opponent is close to the ball, the player may attack the opponent with a gun to try and drain his health to stop him from scoring a goal.



CHAPTER 4 – IMPLEMENTATION

How it works

What is prefab

In Unity there is a very useful physics component to detect collision between objects called Collider2D. It is implemented in a UnityEngine.Physics2DModule and offers a lot of functionality to work with a 2D

collision. A Collider has to be attached to a game object and only collides with other objects that have a collider attached to it. There are various types of colliders, Box Collider2D, Circle Collider2D, Capsule Collider2D, Composite Collider2D, Polygon Collider2D and many more that work in 2D or 3D, like Sphere Collider. Colliders have few basic properties like the gameObject, tag, transform, hideFlags and name.

Walls

To 'build' the walls in the main game area, a number of Box Colliders have been attached to the Football Pitch game object and placed around the pitch. PICTURE. A default setting for a collider is that when another object collides with it, the other object will bounce back if it has a physics component called Rigidbody 2D attached to it which gives the game object control of the physics engine. This means that the game object will be affected by forces controlled from scripts and by gravity. Since this is a top down game, objects should not be affected by the gravity since there is no real up or down but they should be affected by forces. Going back to the colliders, another setting for them is to act like a trigger. This means that whenever another object collides with it, it will 'pass' through it but a message will be sent in a Collision2D format that includes all the information about the colliding object. PICTURE. The second type of collider has been added to a bullet prefab and a basic script has been added which destroys the prefab whenever it hits another object. These walls are very important because they also stop the player and the ball from moving further than the allowed game area. Circle colliders had to be added in the corners of the walls because there was a bug where a ball would sometimes stop just next to the wall which would prevent it from bouncing off it to the side, so it would get stuck and only be able to move alongside the wall. Circle colliders helped to overcome this bug in a way that whenever the ball reaches the corner it would collide with the circle which would bounce the ball away to the side. PICTURE.

The goals

The play area does not have any other components than the sprite which is just a rectangle with lines around it and one line in the middle. The goals are very similar to the playing field but they are made as separate objects they needed to have different tags so it would be easier to write a method that recognizes which goal the ball has hit and which team should get a point. At first it was planned that the goal would be just a straight line and whenever a ball would collide with it, it would count as a goal. However, after testing it turned out this is not the most effective and fair placement of the goals so new goals had to be designed. The new goal allow the player to get inside it similarly like in the game Rocket League, in that game a goal is only counted when the entire ball enter the goal. This allows the players to defend the goal more easily.

The ball

The ball is a more complex object than the wall because there is a bit more information and functionality needed to make it work. First of all, there is a basic Circle Collider2D set up just like in the walls, and the Rigidbody 2D. The most important features here are the material of the object, mass and

linear drag. Gravity can be ignored since as mentioned above, the gravity should not affect any of the objects since there is no 'real' up or down. The material of the object defines its friction and the bounciness. Since it is a ball, its friction should be very small and bounciness should be high, in this case friction is set to 0.1 (or 10%) and bounciness 1 which is a default setting. These setting allow it to behave just like a real ball. Linear drag has been set up with the mass, the higher the mass the harder it is to push the ball therefore the mass had to be kept relatively small. However the side effect of that was that the ball would travel too fast so a linear drag had to be set up, this required few hours of testing and playing around and the final outcome was 0.4 (40%) linear drag. The numbers here are very ambiguous but the most important parts are that friction, bounciness, mass and linear drag had all been used to fine-tune the way the ball behaves.

Another important part of the ball is the 'ball controller' script which keeps track of the goals and updates

Players

Players object are the most complex because there are much more information needed to be passed through it and it has more functionality that is needed to make it work. Just like the walls and the ball it has all the components that take care of the collision and movement. Since the sprites are circular, Circle Collision2D components have been used to make give the best performance possible. Complex collision detection may sometimes be processor heavy which can lead to slow performance and bugs. Rigidbody 2D has also been used but this time it was much more straightforward to set up because player has to move only when the button is pressed or when the player tilts the joystick stick in a certain direction. The player does not stop immediately when a player stops pressing a button but it still stops relatively fast compared to the ball which bounces around. If a slow slow stop time and a high mass were set up it would make the player's character feel like it is sliding on ice which is not the desired effect. The game is fast paced therefore it makes most sense to make the controls feel very responsive and easy to control. Therefore it took a lot of tests to come up with the best settings for the player movement.

What is start, update, fixed update

Values

Controls

A player object can be controlled in two main ways; by Artificial Intelligence or by a joystick/keyboard. This is controlled by a Boolean value called 'aiControlled'. If that value is true, the Update method will not be processed and the controls will be automatically picked up by the AI game object. Joystick controls are also split into two, first is that the player controls the movement with left analogue stick and the rotation with the right, the second way is that the player controls the movement and the rotation with the left stick. The first way is how the game was meant to be controlled initially because it

is similar to the way other popular shooter games control, left stick is used to move and right stick is used to point the gun which allows for very flexible controls. At first this seemed right but after adding more features it turned out that right hand had to also be used for shooting, shield, attacking, changing guns and reloading. In a fast paced game this can turn out to be very difficult to manage and creates a very steep learning curve. After playing other games like Grand Theft Auto 2 and Rocket League, a new way has been added that is easier for the new players. In GTA players control the movement and the rotation with the left stick which leaves the right hand for other task. In a 3D game, Rocket League, players can switch between having the camera pointing at the ball all the time or having a classic Behind The Player camera movement. Having the camera pointing at the main point of focus makes the game a lot easier to control but it does not allow to easily look around the game area which can sometimes be crucial to the outcome of the game. Players can easily switch between two modes which adds flexibility and fits much more playstyles. This project uses both methods and just like in Rocket League, players can switch between the two control types so new players with rotation and movement in the left stick and once they gain more skill, they can switch to the other way for more flexible controls.

Keyboard controls

Rotation works in a fairly basic way, an angle is calculated by first finding the arctangent of the x,y coordinates then by converting it to degrees using the Mathf.Rad2Deg method and then by converting it to a Quaternion rotations using the AngleAxis method and lastly by using the Quaternion.Slerp method which by the definition in Unity manual, spherically interpolates between two points by using a time parameter. Quaternion.Slerp can be really useful because the speed in which the player rotates can be adjusted so the game looks and feels much smoother than if the player object was ‘snapped’ into the wanted position.

Movement, compared to rotation is very straightforward; RigidBody2D objects have a method called AddForce which is used to add force to an object continuously. The first part of the movement method checks if the movement is in the deadzone, if the joystick is only tilted very lightly there is no need to move the player object because this can mean that the joysticks rest position is not exactly 0, 0 but can be something like 0.013, 0.09. The deadzone value can be adjusted for a best feel but it should not be high because this could make the player movement feel disjointed.

The sprint is a very basic method which checks the player’s stamina status and if the player has enough stamina, it changes the speed float value by adding the sprint power which is stored in the global settings. It also drains the player’s stamina level.

Weapons

Pickups

Wall

Behaviour

The Shoot method in the weapon game object is exactly that, the state is first checked and if it is idle, player is allowed to shoot. Which in reality takes the AbstractGun variable “currentGun” which runs a shoot method inside it.

AbstractGun is an abstract class which inherits the MonoBehaviour base class and the IGun interface. It holds all of the default functionality of a basic gun which does not need to be overridden. New guns inherit this class mainly to make the code easier to understand but it could be just as easily used as the main class for the weapons. The shoot method checks if there is enough bullets, if there are not it will call the Reload method and the Shoot method again, and if it has enough bullets it will call a Fire() method in the shooter class which takes care of Instantiating bullets.

Shooter class has been added to be able to instantiate bullets in a random places to add more reality to the game and if there will be new variables like for example, wind that will affect the bullets, shooter class will be able to take care of that without having to add new functionality to the Shoot method. As one of the SOLID principles says, ‘Methods should only have one responsibility’.

A reload method is very basic, it checks if player has enough bullets and adds them to the magazine accordingly. It also runs a DelayMethod(reloadDelayTime) which delays when the player is able to shoot for a given time. The reload time adds customizability to the weapons because each can have its own reload time.

Describe how the implementation maps onto the design you have already discussed.

You should use “code snippets” to illustrate special features of your work or difficult (awkward) bits of coding. Don’t make any of these snippets longer than half a page (and include line numbers if possible). If the code fragment is longer than half a page then break it up into smaller bits.

Describe the code, both in terms of the overall architecture and in terms of the snippets. Make sure the reader understands what you have done and why!

CHAPTER 5 – EXPERIMENTATION AND EVALUATION

Again, what it says on the tin!

If you haven't done too much testing (for instance it is GUI based) then include a "walk-through" of the application with screenshots showing the scenarios in which the application can be used. After all, the examiners may not be near a computer to actually "run" your code.

You also need to clearly show how you have evaluated your work and show how it meets the original aims and objectives.

CHAPTER 6 – CONCLUSIONS

What have you learnt?

Basically, “reflect” on the work you have done.

What additional features/extensions can be done to the work and/or what would you have done if you had more time.

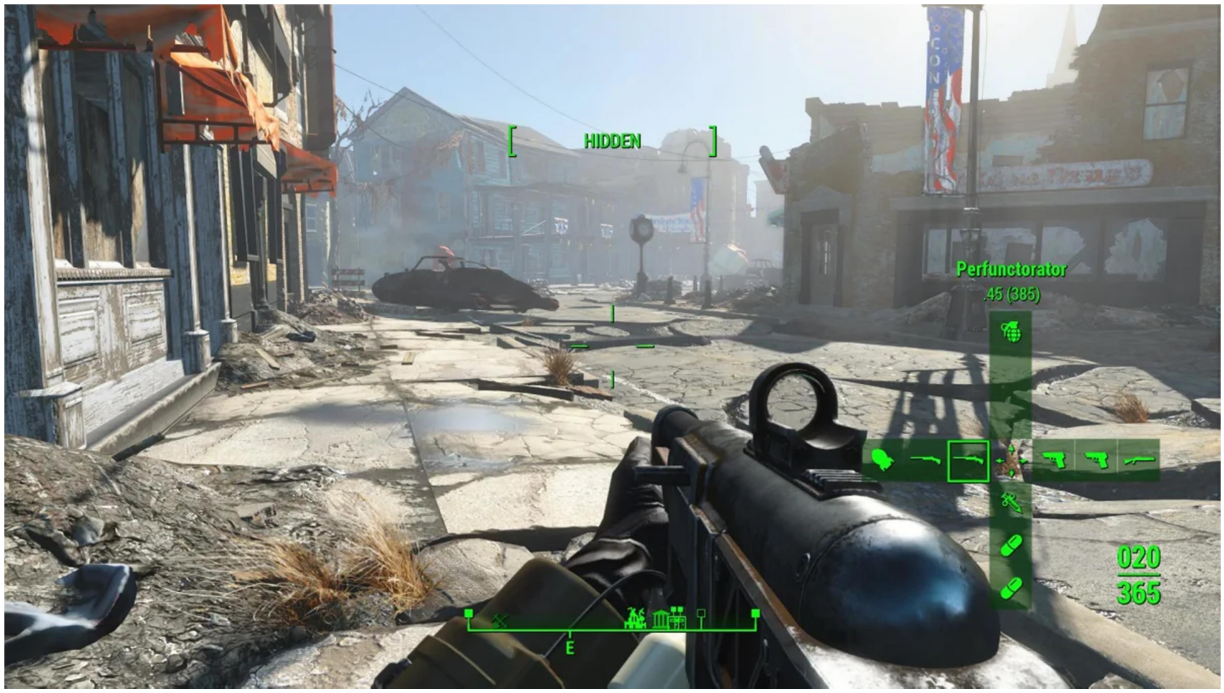
APPENDICES

Include design diagrams, data formats, etc.

Basically, don’t overfill the report.

Link to the GitHub repository

Game UI




4:54 AM / CLEAR
THE FALL OF THE HOUSE OF REARDON
Kill the monsters in the manor.
Warg
5 Viziman Champion
10 Swallow
35 Blunt crossbow bolt


4:00
YOU
PACKET LOSS
INDICATOR
100
BOOST


DIVISION RIVALS
28:36
AVE | 1-1 | Ban
R1 CALL FOR 2ND PLAYER
R2 CHANGE KICK TAKER
L1 SWITCH RECEIVER
L3 CHANGE CAMERA
R ADJUST START POSITION
TARGET ON/OFF
7 ROBERTO FIRMINO
94 FABINHO
START


$00048624
16:45
193 13
POLICE

REFERENCES