Faculty of Science and Technology

BSc (Hons) Games Technology

May 2018

Design, Implementation and Testing of a Real-Time Strategy Game

by

Jake Ruggier

DISSERTATION DECLARATION

This Dissertation/Project Report is submitted in partial fulfilment of the requirements for an honours degree at Bournemouth University. I declare that this Dissertation/ Project Report is my own work and that it does not contravene any academic offence as specified in the University's regulations.

Retention

I agree that, should the University wish to retain it for reference purposes, a copy of my Dissertation/Project Report may be held by Bournemouth University normally for a period of 3 academic years. I understand that my Dissertation/Project Report may be destroyed once the retention period has expired. I am also aware that the University does not guarantee to retain this Dissertation/Project Report for any length of time (if at all) and that I have been advised to retain a copy for my future reference.

Confidentiality

I confirm that this Dissertation/Project Report does not contain information of a commercial or confidential nature or include personal information other than that which would normally be in the public domain unless the relevant permissions have been obtained. In particular, any information which identifies a particular individual's religious or political beliefs, information relating to their health, ethnicity, criminal history or personal life has been anonymised unless permission for its publication has been granted from the person to whom it relates.

Copyright

The copyright for this dissertation remains with me.

Requests for Information

I agree that this Dissertation/Project Report may be made available as the result of a request for information under the Freedom of Information Act.

Signed: 

Name: Jake Ruggier

Date: 17/05/2018

Programme: BSc Games Technology

ACKNOWLEDGEMENTS

I would like to thank my primary and secondary supervisors, José Fonseca and Glyn Hadley, along with all those who took part in and submitted feedback for the testing phase of this project.

ABSTRACT

The aim of this project was to design and develop a casual real-time strategy game in order to study key aspects such as unit balance the work required to successfully implement them, as well as how a game of this kind may keep players entertained with an example of emergence that comes with the application of a few simple rules to objects in a scene.

A study of the state of the art, including game design theory and examples of existing games of the genre and of emergence, namely Conway's Game of Life, informed the design of the game. A complete development followed with the creation of 3D assets, programming and game engine implementation. A testing phase sought to ensure that the game was well balanced, iron out any bugs and form a conclusion about the success of the game.

This has culminated in the creation of *WWII Simulator*, a stylised, light-hearted take on World War II battles in the form of a simple real-time strategy game.

FIGURES

CONTENTS

1. INTRODUCTION, AIMS AND OBJECTIVES

This project aims to demonstrate the importance of balance in video games, specifically strategy games, and explore the process of achieving it. It will also examine how the concept of emergence may be applied to a game with large numbers of elements with basic rules, leading to an entertaining and fulfilling experience for players.

To do so, a complete real-time strategy (RTS) game will be developed for the PC. RTS games make up a subgenre of strategy video games. Events in RTS games progress over time, as opposed to turn-based strategy games which progress turn-by-turn as a board game would. They tend to take the form of large game maps in which the player has control of a number of elements, such as groups of soldiers. They may have a choice of what element they get to control, and may be able to instruct them to, for example, move to a location and battle with an enemy.

The proposed game for this project will be a casual, stylised RTS game with a Second World War setting, titled *WWII Simulator*. As well as creating the game with behaviours for different units, there will a strong emphasis on ensuring they are well balanced, i.e. that none is too weak or too powerful, through testing and adjusting of their characteristics.

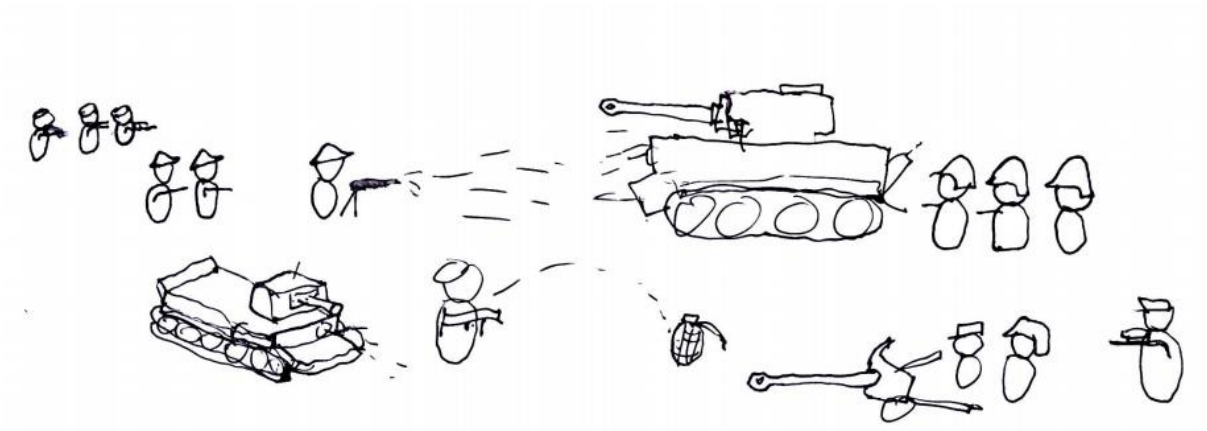


Figure 1. Early concept sketch for WWII Simulator

This aim will require first consulting literature such as game design theory to inform the development of the game and the project as a whole. The development will then start with the formalising of the design for the game, followed by creation of all required assets, programming and implementation in a game engine. There will then be a phase of testing and refining the game, making the use of volunteers in order to identify where the game has succeeded and what changes must be made to improve it, particularly in the area of game balance.

2. STATE OF THE ART/LITERATURE REVIEW

This section will look at existing games of the genre and literature on game design theory to inform this project. It will also study Conway's Game of Life and how an interpretation of emergent events as entertaining scenarios could also be applied to this type of game.

2.1 GAME DESIGN THEORY

Books and videos on game design were used to guide the design and development of the game itself. As well as theories and tips on game design in general, there was a focus on studying sections specific to the strategy genre and the particular issues that come with them, such as game balance.

Chapter 11 of *The Art of Game Design* (Schell 2014) covers the types of game balance and the difference ways of achieving them, as well as how important and rewarding a part of the game development process it is. It describes it as "the most artful part of game design, for is all about understanding subtle nuances in the relationships between the elements of your game and knowing which ones to alter, how much to alter them, and which ones to leave alone". This is a key justification of the decision to focus on game balance in this project.

The game will be 'asymmetrical', meaning, rather than units simply all being the same to ensure a level playing field, they will have different strengths and weaknesses balanced against each other. For example, one unit might be weaker in terms of maximum health/durability, but make up for it with higher accuracy or rate of fire. The differences between units, and the different possible strategies they bring about, will make pitting different forces against each other 'interesting and thought-provoking', and be one of the key features to make the game fun.

One approach for achieving asymmetric balance is described by Schell with the analogy of 'Rock, Paper, Scissors'. "None of the elements can be supreme, because there is always another than can defeat it". This approach allows certain units to be more powerful, as long as there is a strong counter to it, a clear strategy against which its strengths become obsolete. For example, tanks may be very strong against infantry and machine guns, but if up against them, the player can simply adjust their strategy and use plenty of anti-tank guns, neutralising that imbalance. This concept is described in an episode of game design video lesson series *Extra Credits* (2012) as 'perfect imbalance'. It argues that having some slight imbalances make games a lot more engaging, as it means the player has to figure out the right strategy to counter certain units/weapons which makes them weak when employed against them. That strategy itself will have weaknesses in other scenarios. If there are no strategies which can't be countered by anything, then the game can be considered well-balanced.

Other general game development advice given by Schell and others which would be useful in the development of *WWII Simulator* is that sometimes, removing/cancelling content

rather than adding it is the right option to make the game as good as it can be. As Scott Rogers advises in *Level Up!: The Guide to Great Video Game Design* (2010), “Don’t become so attached to your ideas that you lose objectivity”.

2.2 EXISTING GAMES

There are numerous established real-time strategy series with large followings. One such series is *Company of Heroes* (Relic Entertainment 2006). The original is widely considered to be the one of the best real-time strategy game set in the Second World War, described in *Kotaku* (Plunkett 2016) as the “pinnacle of the genre itself”. It features several playable nations with historical soldiers, weapons and vehicles.

Over the course of a game of *Company of Heroes*, the player must unlock, build and upgrade different units, consisting of groups of soldiers of individual vehicles, and use them to defeat the enemy. They have full control of each unit’s movement around the game map, sending them to attack particular enemies, capture territory or simply to defend from a spot with lots of cover. They can choose what units to invest most in depending on their strategy, but will have to adapt to how their opponent plays. For example, more anti-tank weapons will be needed if they attack with a lot of vehicles.

Figure 2. Screenshot of Company of Heroes gameplay

Totally Accurate Battle Simulator (Landfall Games 2016) is a very different type of strategy game. It is a smaller, independent early-access game with a medieval setting. As opposed to the realism of *Company of Heroes*, the game is very stylised and much more casual. In each level, the player has a budget to spend on various low-polygon humanoid soldiers which run into and attack a set arrangement of enemy soldiers after pressing play.

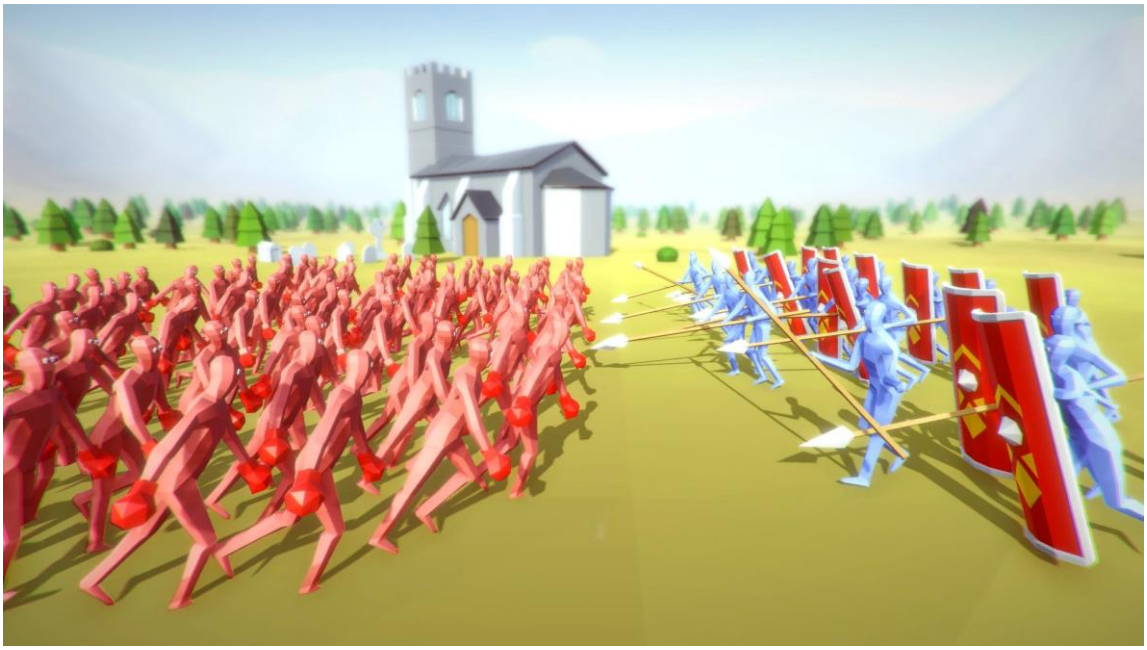


Figure 3. Gameplay screenshot of Totally Accurate Battle Simulator

Rather than being realistic simulation of battle or deep tactical experience, the unique selling point of Totally Accurate Battle Simulator is the fun of watching your choice of such strangely-portrayed characters flail about in the game's surreal representation of combat.

This has made given the game viral popularity through YouTube. One video, uploaded by 'FGTeeV' (2017), has accumulated over 9.2 million views.

2.3 GAME OF LIFE

Emergence is described as what occurs when the whole is greater than the sum of the parts. Examples of it can be seen in nature, such as the formation of the fractal shapes of snowflakes. Starlings create fantastic murmurations when flying in a large flock. Termites, despite individually being very simple creatures with no concept of how to design a structure, can create large, complicated nests when working as a colony of thousands by simply following chemical cues left by other termites (Burbeck 2004).

This project won't be simply drawing from game design theory and existing games to make a good real-time strategy game. It will be looking into this phenomenon of complexity arising from a few simple rules and how that can be seen to take the form of emergent narrative which can enrich this type of game. Mathematician John Conway's Game of 'Life',

a cellular automaton, demonstrated the concept when it was published in Martin Gardner's column in Scientific American (1970). It was named as such because it served as a mathematical explanation to how the basic rules which govern the universe can lead to the emergence of complex organisms.

The game (Conway describes it as a 'no-player game' (Numberphile 2014)) consists of a grid of square cells, each of which can be either 'live' or 'dead'. What state they're in is determined by four rules:

1. If a dead cell has exactly 3 live neighbours (defined as one of the 8 surrounding cells, both adjacent and diagonal), it becomes live.
2. If a live cell has fewer than 2 live neighbours, it dies, as if by underpopulation.
3. If a cell has more than 3 live neighbours, it dies, as if by overpopulation.
4. If a live cell has 2 or 3 live neighbours, it survives.

With just these four rules governing how the cells change each cycle, complex and fascinating formations can emerge from different starting arrangements, grow, move around and replicate, and either 'die out' or become static after a few cycles, after many cycles or carry on forever. Some arrangements have become well known, such as Gosper's glider gun (*Figure 4*), which produces an endless stream of 'glider' patterns which move in a diagonal direction as they repeatedly transition between 4 different shapes of 5 live cells. The glider gun can even be used to create logic gates within the *Game of Life*, meaning it can function as a Turing machine. Others appear random, like explosions or bacterial colonies, with no mathematical way to predict for certain whether they will eventually die out.

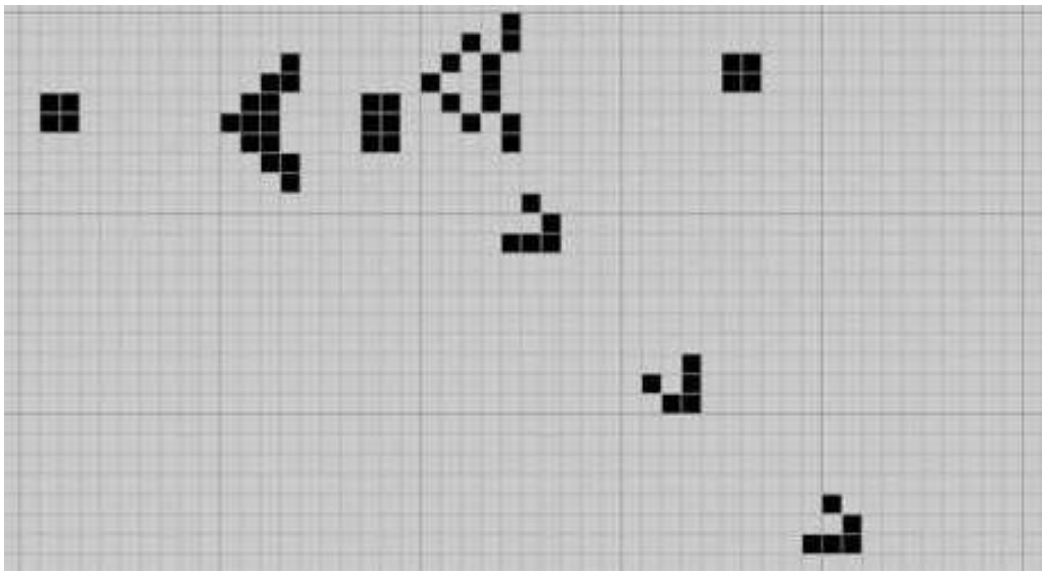


Figure 4. A computer implementation of Conway's Game of Life with a recreation of Gosper's glider gun

Though it was originally created without computers, using counters on a *Go* board, many people have since programmed it and there are numerous websites on which the *Game of Life* can be played. It can be a fascinating and addictive experience, testing different arrangements or just adding individual cells and seeing what happens. It could result in large, beautiful movements which spread across the grid, or in a fast destruction of itself. Some of the formations people have created over the years could be describes as artistic, and are so enjoyable to watch that one video of various large-scale Life scenarios titled 'epic conway's game of life' (sic) uploaded by YouTube user Emanuele Ascani (2011), for example, has over 2.1 million views.

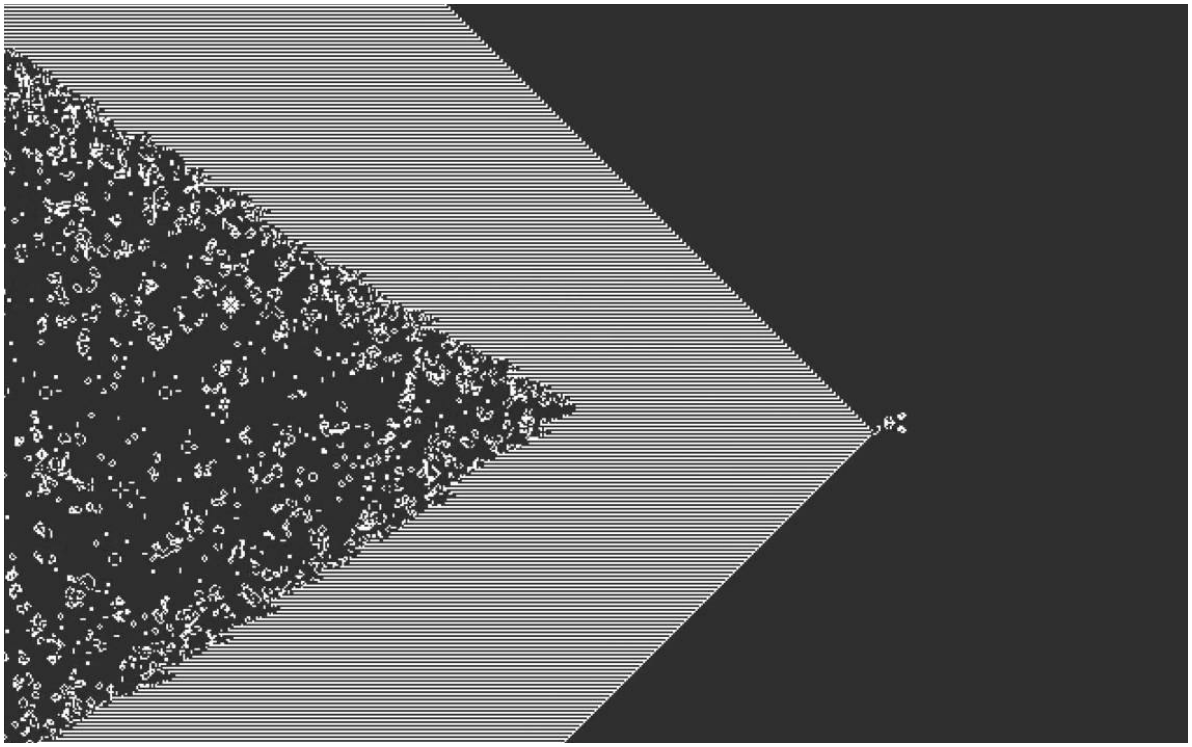


Figure 5. An example of one of the large-scale formations that have been created in Conway's Game of Life featured in Ascani's video

This project will attempt to create a similar experience using the format of a real-time strategy game, in terms of a few simple unit behaviours programmed being turned into enjoyable-to-watch emergent events when arranged into different large-scale starting combinations.

3. DEVELOPMENT

This section will describe the methodology and development process for the game. This included the design, 3D modelling, programming, game engine implementation and testing.

3.1 RESEARCH AND DESIGN

The first step was the research to inform the design and implementation of the game, including the study of game design theory and other games as covered in sections 1 and 2.

Both *Company of Heroes* and *Totally Accurate Battle Simulator* were drawn from in the design of the game. The basic template for the gameplay, as well as the visual style and humour, owe more to the latter.

WWII Simulator shares with *Totally Accurate Battle Simulator* the system of placing different units into a scene limited by a budget and watching them fight. The enjoyment you can get from simply watching the battles based on the units you put in, with the comical simplicity of the look and behaviour of the soldiers, was a key inspiration. As Casey Chan put it in *Gizmodo* (2016), “the beauty is in the hilarity”.

WWII Simulator’s own version of this hilarity and simplicity is a justification for the early decision to limit the player’s agency to affect the battle to the selection and placement of units, after which they would just watch. This works because of the key objective of the project to make the battles really enjoyable to watch, as it is in *Totally Accurate Battle Simulator*. Players laugh at the way the soldiers move and attack each other, they find themselves rooting for the last surviving soldiers on their side, and they always want to try out new combinations, not just for the motive of winning, but for the desire to see how they play out.

But to make it work as a strategy game, even a casual one, making the choice and placement of units the sole means of the player to affect the outcome of each battle requires a lot of thought to be put into these unit types. They each need to be very distinctive and have very clear, possibly exaggerated, strengths and weaknesses.

The influence of *Company of Heroes* is largely in the World War II setting and the different unit types that come with it and, importantly, how they complement and counteract each other. Each has its role. Tanks support infantry, in particular against machine guns and other vehicles. But they will generally struggle without infantry, which in turn supports them against anti-tank units. Units like snipers aren’t to be used to attack the enemy on their own, but can effectively support other units from a distance. Some units can be produced in greater numbers, others are more powerful but more costly to produce.

The plan from the start was to keep the same emphasis on this dynamic, on the importance of specific units being required alongside others to tackle specific challenged, with room for alternative strategies for players to experiment with.

It was decided that there would be 7 unit types to feature in *WWII Simulator*:

Infantry – Basic soldier unit with rifle.

Sniper – Can't take much damage, keeps distance and fires accurate shots, prioritising machine gunners and other snipers.

Grenadier – Tries to get around enemy tanks to fire their low-penetration handheld AT weapon into the thinner side and rear armour. Also has a grenade to throw.

Machine gun – Places itself at a distance from the enemy. Very rapid but inaccurate fire is highly effective against groups of enemy soldiers.

Anti-tank gun – Keeps its distance and fires high-penetration shells at tanks, or other units if necessary, loses effectiveness at short range.

Medium tank – Fast moving, lightly armoured tank. Will attempt to flank enemy heavy tanks. Immune to bullet fire, and very effective against soldiers and emplacements.

Heavy tank – Slower tank with stronger frontal and side armour. Grenadiers and medium tanks will have a harder time penetrating it.

Another aspect of *Company of Heroes* to emulate was the appeal of the playable nations' different 'characters', in terms of their visuals as well as how they play. Each has its own recognisable vehicles, weapons and uniforms for different soldiers, and also its strengths and weaknesses based on how it fought in the war. Senior designer Quinn Duffy described in an interview with *PC Gamer* (2016) that they developed each army's 'essence' by looking at what they did historically to lead to their successes, then "pitting them against each other again and again and tweaking and tuning".

An important part of *WWII Simulator* was making each nation's soldiers and each of their weapons and vehicles recognisable. For each, the most appropriate one to represent had to be determined, based on their fame and historical significance, balanced with what their role would be in the game. This required some research.

There were some iconic helmets and hats worn by the different countries that fought in the war, and it was decided that two would be represented for each in the game (one for most soldiers, the other for a few specialised soldiers). For the British, it was concluded that the two most recognisable were the wide-rimmed Brodie helmet, "the most instantly recognisable symbol of the British Tommy" (Pegler 1996), and the beret. For the Germans, the primary headgear would be their equivalent helmet for the two world wars, the Stahlhelm, and the secondary would be based on the caps like those worn by the S.S.

Figure 6. Colourised photograph of a British soldier with a captured young German soldier in central Italy, taken by C. Bowman in 1944, colourised by R. Mallow (2017)

The most common other weapons of the same ability for each nation were researched, deciding on:

Machine gun: British Vickers MG, German MG-42.

Grenade: British Mills bomb, German Stielhandgranate.

Handheld AT launcher: British PIAT launcher, German Panzerschreck.

Anti-tank gun: British 17-pounder. German 8,8cm Pak 43.

The same was then done for tanks. The Cromwell, which saw extensive use by the British Army in the latter stages of the war, is the perfect fit for a fast tank with relatively weak anti-tank capability. The Panzer IV, while having similar armour characteristics, is known more for its anti-tank gun than its mobility, but it still appropriate to fill the role as Germany's medium tank. It would be important to ensure that neither performs better in-game that the other and would either share the value for aspects like armour, speed and penetration for the sake of the game, or have any differences balanced against each other. Britain's heavy tank series in the war was the Churchill, and the Mk. III would be their representative in the game. The German heavy tank could be nothing other than the Tiger.

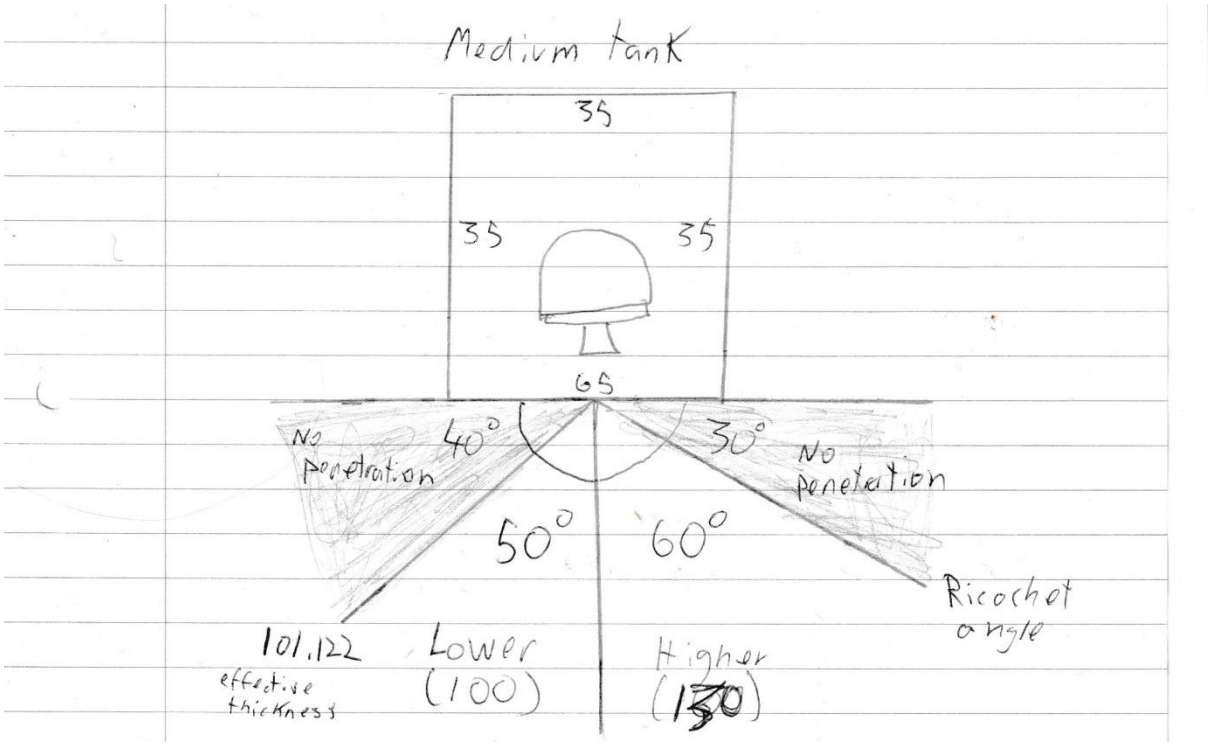


Figure 7. Design sketch and values for armour thickness and penetration, showing the range of angles within which certain shells will penetrate the front of a medium tank.

Decisions were made for the values of front and side armour thickness and of penetration ability of heavy tanks and AT guns, medium tanks and grenadier handheld AT launchers by calculating what it would take to penetrate in different scenarios at different angles (*Figure 7*).

All of these design decisions were put into a game design document (Appendix A), along with details of the game mechanics, asset lists and unique selling points. It is important to define at the start the scope of games like this to avoid ending up with either a lack of features, or 'feature creep', when features keep getting added midway through development, making it impossible to keep to a deadline (Greene 2017).

3.2 ASSET CREATION

Nearly all of the assets to be made were 3D models. They needed to be made for the head, body and hats/helmets of soldiers, weapons including rifles, grenades and anti-tank guns, and tanks.

The software used for all of this modelling was *Maya LT 2016* (Autodesk 2015). It was chosen over more complex sculpting software packages like *Zbrush 4R7* (Pixologic 2015) because the objects to be modelled, with the simplistic art style, will only require hard-surface modelling which *Maya* is designed for, rather than organic models. Using *Maya* also

allowed the polygon count to be kept as low as possible. Maximising performance is important for a game in which large numbers of game objects may be in play at once.

To ensure the dimensions of the models were accurate, blueprints from *the-blueprints.com* (EVOlution Graphics 2009) were used for each of them, with up to 3 image planes and the orthographic camera, as well as photos of the subjects.

A stylised, cartoon-like art style had been chosen for the game, so no time needed to be spent on texturing the models in *Maya*. They would simply have block-colour materials applied to them in *Unity*.

It was important to make sure that certain parts of models like the tanks were kept as separate meshes (but still saved together) if they needed to move/rotate separately (e.g. turret) or have a different material (e.g. tracks).

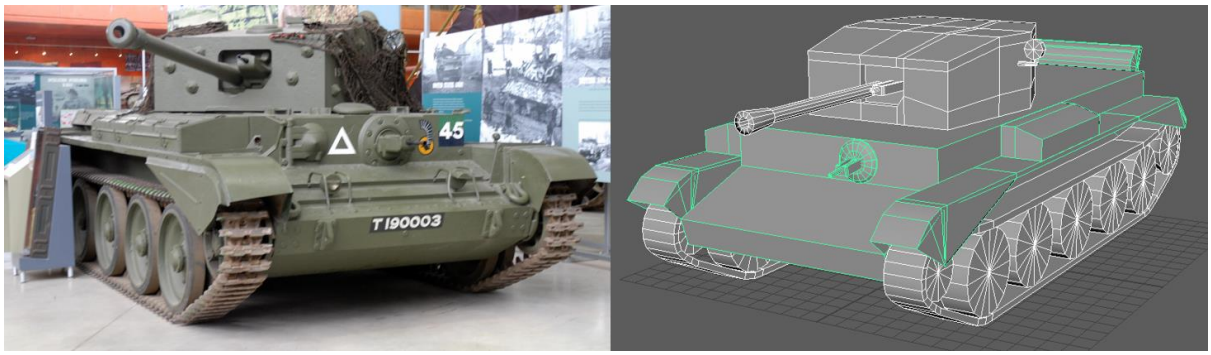


Figure 8. Photograph of Cromwell tank at Bovington Tank Museum in Dorset, UK, alongside screenshot of the WWII Simulator model in Maya

To fit with the art style and keep polygon counts low (for the reasons highlighted above), the models are all simplified representations of the real objects/vehicles, excluding any small details which have no relevance to the gameplay. For example, in the model for the Cromwell tank, only the basic shape of the front hull is modelled apart from the hull machine gun, as that's something which is functional in the game.

3.3 IMPLEMENTATION/PROGRAMMING

The game was made in *Unity 5* (Unity Technologies 2015), a multi-platform game engine which allows 3D games to be made far more quickly than creating an engine from scratch. It has an in-built physics system, which was to be very useful in *WWII Simulator*. All scripting was done in the C# object-oriented programming language, using the *Visual Studio 2015* IDE (Microsoft 2015).

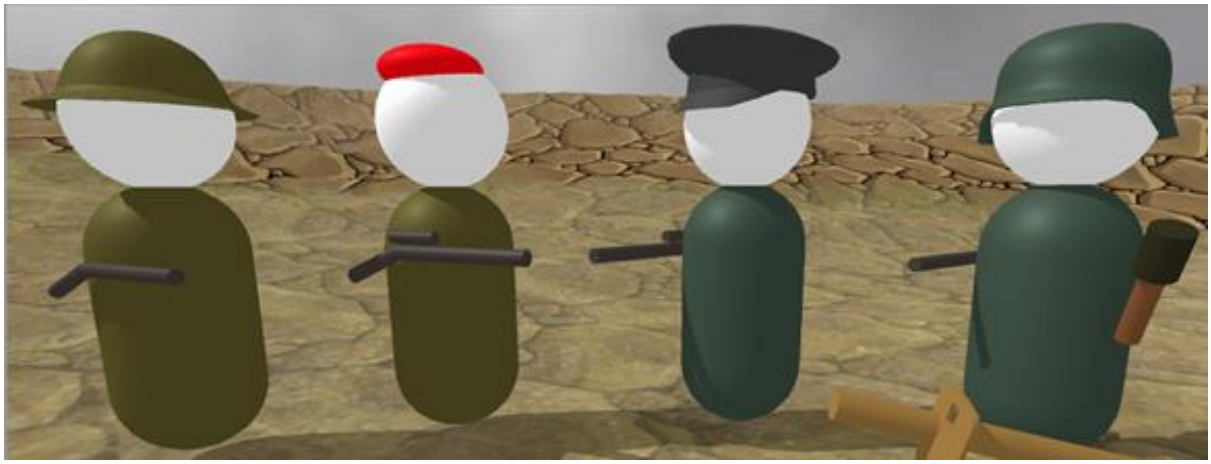


Figure 9. British and German soldier models in Unity with basic materials applied

The first task in *Unity* was to finalise the look of the models in-game. After importing the models for the soldier head, body, helmets and hats, they were put together as child objects of parent 'Tommy' and 'Jerry' soldier objects and saved as prefabs. Basic colour materials were applied for the clothes and headwear (*Figure 9)*. After experimenting with realistic skin colour materials, it was decided that they would be used rather than a basic white mannequin colour for all of them. British soldiers would have a larger variety of possible skin tones, representing the men from across the Empire and Commonwealth who fought in the war, compared the mostly white forces of Germany.

An asset from the Unity Asset Store, UniTOON Ultra (Warhead-Designz 2015) was then used to give the textures a cel-shaded look, as well as thick outlines, making the game objects look like part of a cartoon. The other models, including the tanks, were then imported, turned into prefabs and had their materials applied.

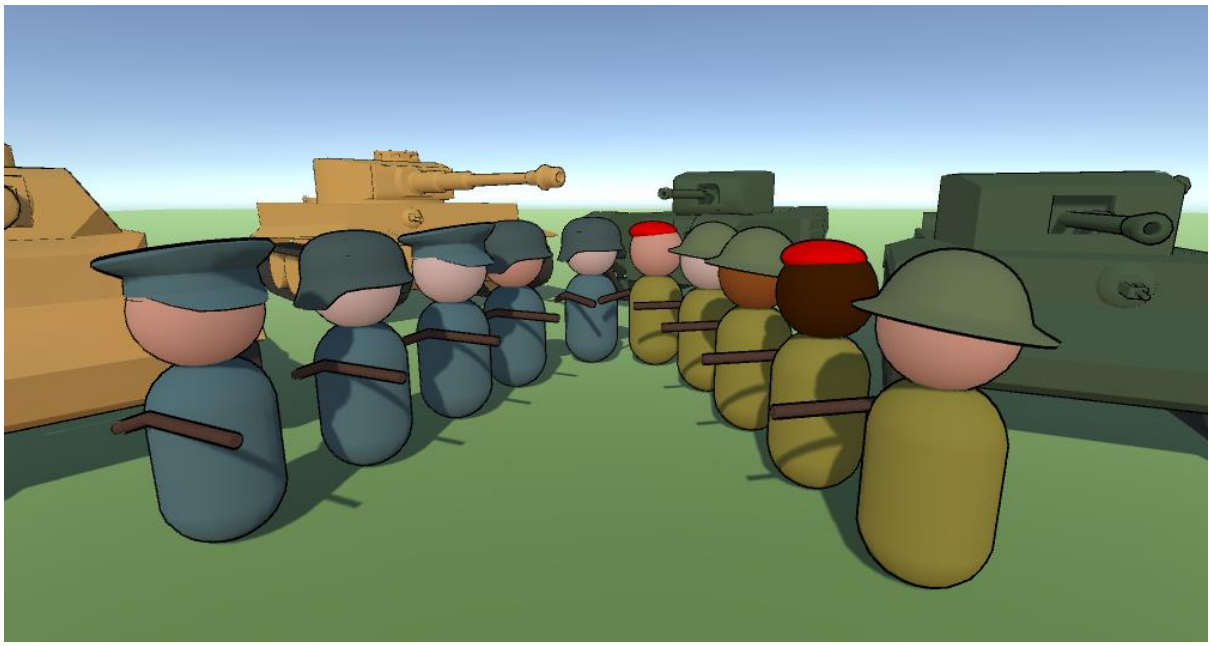


Figure 10. Unity screenshot of finalised soldiers with 'cartoon' materials applied

The tank and heavy weapon prefabs in particular had to be put together in terms of parent and child transforms in a specific way. For example, the tank turrets must be parents of an empty 'TurretPivot' object, itself a child of the hull object and placed at the point around which the turret was to rotate.

Work then started on programming, which was to use object-oriented techniques. All units are objects of classes inheriting from a 'Unit' parent class (*Figure 11)*. It contains variables and methods which all units share, such as the health, rate of fire, an array of all enemies' scripts and functions for calculating the closest enemy. Most are protected, meaning they can be accessed from inheriting scripts but not those of other objects. Some, including the team, unit type and the function for determining units within an explosion's radius are public as they need to be accessed by other objects' scripts. There are also virtual methods like those for moving and attacking, which all units have, but each defining them differently using an override in their respective derived classes.

```
public class Unit : MonoBehaviour {

    public int type;
    public string team, enemyTeam;
    public bool dead;
    public float health;
    protected float speed, lookSpeed, rateOfFire;
    protected float variation = 0.3f;
    protected float maxFireDistance, stoppingDistance;
    protected float upperROF, lowerROF;
    public Rigidbody rb;
    protected GameObject[] enemies, allies;
    protected Unit[] enemyScripts, allyScripts;
    protected bool gameOver = false;
    protected string priorityTarget;
    protected bool sightBlocked = false;
    public Transform leftMarker, rightMarker, rearMarker;

    public enum Type { infantry, sniper, grenadier, machinegun, atgun, heavytank, mediumtank };

    protected virtual void Movement(Transform targetTransform, int targetIndex) { }
    protected virtual void Attack(Transform targetTransform, float targetDistance, int targetIndex) { }
    protected virtual void Strafe(Transform closestEnemy, Transform obstacle) { }
    protected virtual void MachineGunOrGrenade(Transform targetTransform, float targetDistance, int targetIndex) { }
    public virtual void hitByExplosion(Vector3 explosionPos, float explosionRadius, float explosionForce, float distance) { }

    protected void Teams()...

    protected void Targeting()
    {
        Transform targetTransform = enemies[0].transform;
        float targetDistance = 0;
        int targetIndex = 0;
        ClosestEnemy(2000, "all", ref targetTransform, ref targetDistance, ref targetIndex);
        if (type == 2 || type == 5 || type == 6) { MachineGunOrGrenade(targetTransform, targetDistance, targetIndex); }
        ClosestEnemy(maxFireDistance, priorityTarget, ref targetTransform, ref targetDistance, ref targetIndex);

        if ((type == (int)Type.grenadier && enemyScripts[targetIndex].type >= 5) || (type == (int)Type.mediumtank && enemyScripts[targetIndex].type == (int)Type.heavytank))
        { Movement(closestMarker(targetIndex), targetIndex); }
        else { if (type == (int)Type.infantry && enemyScripts[targetIndex].type == (int)Type.atgun) { targetTransform = closestMarker(targetIndex); }
               Movement(targetTransform, targetIndex); }
        Attack(targetTransform, targetDistance, targetIndex);
    }

    public void ClosestEnemy(float searchRange, string priorityUnits, ref Transform targetTransform, ref float targetDistance, ref int targetIndex)...
```

Figure 11. A sample of code from the Unit parent class

The 7 base unit type classes each inherit from one of three classes which derive from unit. *Infantry*, *Sniper* and *Grenadier* inherit from *Soldier*, *Machine gun* and *Anti-tank gun* inherit from *Heavy weapon*, and *Heavy Tank* and *Medium tank* inherit from *Tank*. Those within each one share many variables and methods which others don't, as well as having similar Unity transform hierarchies. For example, all soldiers have a weapon in the same place, and all heavy weapon unit have a pivot around which the gun will rotate when it's lifted.

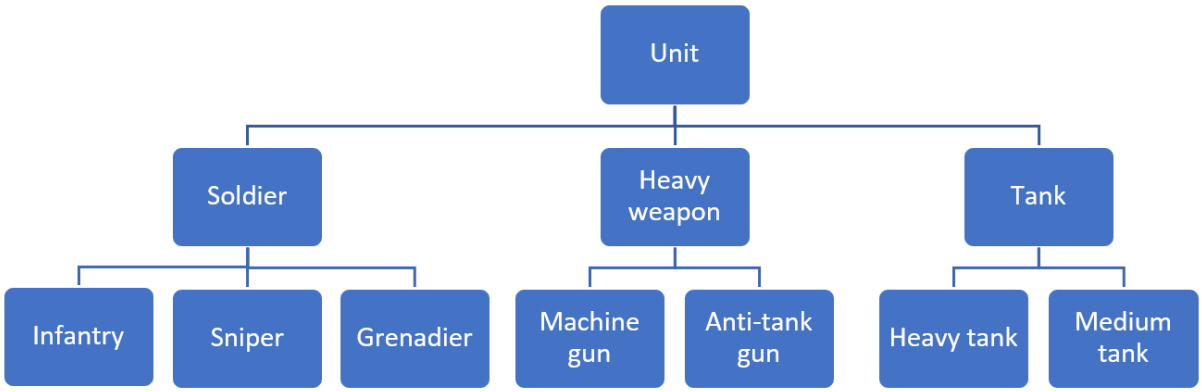


Figure 12. Chart showing hierarchy of class inheritance for units

This whole structure makes the program a lot more efficiently-written, avoiding repeated code. It allowed new units to be programmed quickly once the parent classes had been completed. Many of the child classes, like those for snipers and infantry, contain very few lines of code, requiring little more than different values for variables such as *speed* and *rateOfFire* to be assigned in their class declarations.

The first units to complete were the infantry and the sniper. As mentioned above, they mostly use the same code from the *Unit* and *Soldier* classes and behave very similarly. One key difference is that the sniper stops moving towards an enemy if it is within its maximum firing range, implemented with a simple if statement in the *Movement()* function in the *Soldier* class. Another difference comes with the *priorityTarget* string variable which was implemented. It's used in the *Targeting()* function in the *Unit* class, from which the *Movement()* and *Attack()* functions are called. It calls the *closestEnemy()* function twice; first to find the closest enemy in the scene regardless of unit type, then to find the closest unit of a particular type or types within its firing range. In this case of the sniper, the string value of "snipermachinegun" means that if any enemy snipers or machine gunners are within its range, it will shoot the closest one of them, even if there are infantry units closer.

Work in the Unity editor continued during the scripting process. As the basic functionality for the infantry and snipers took shape, a basic up-and-down animation for them was made to play when they're moving, to give them the look of hopping towards their target, as opposed to simply sliding along the ground. Code was quickly written to turn toggle the animation on and off based on the movement. A random delay in starting the animation at the beginning of a game was also implemented so soldiers don't move up and down in perfect timing with each other, which looked unnatural.

Also created at this point were the particle systems for the bullets of their rifles. It was decided to use particle systems rather than projectile game objects, because the latter have a bigger impact on performance, and are less accurate with the possibility of fast projectiles passing through small colliders without a collision being detected.

The particle system for infantry's bullets was made first. It consists of short yellow trails that get narrower and fainter further back. When one is emitted, it travels at high speed (but less than a realistic speed so it can be seen in-game) in a forward direction, but with random deviation within a narrow cone shape. When the soldier isn't firing its weapon, the emission over time is set to 0. When it is set to start firing, a function in the *Soldier* class sets the emission over time to be a random between two constants, being the upper and lower rates of fire calculated with the unit's *rateOfFire* and *variation* variables.

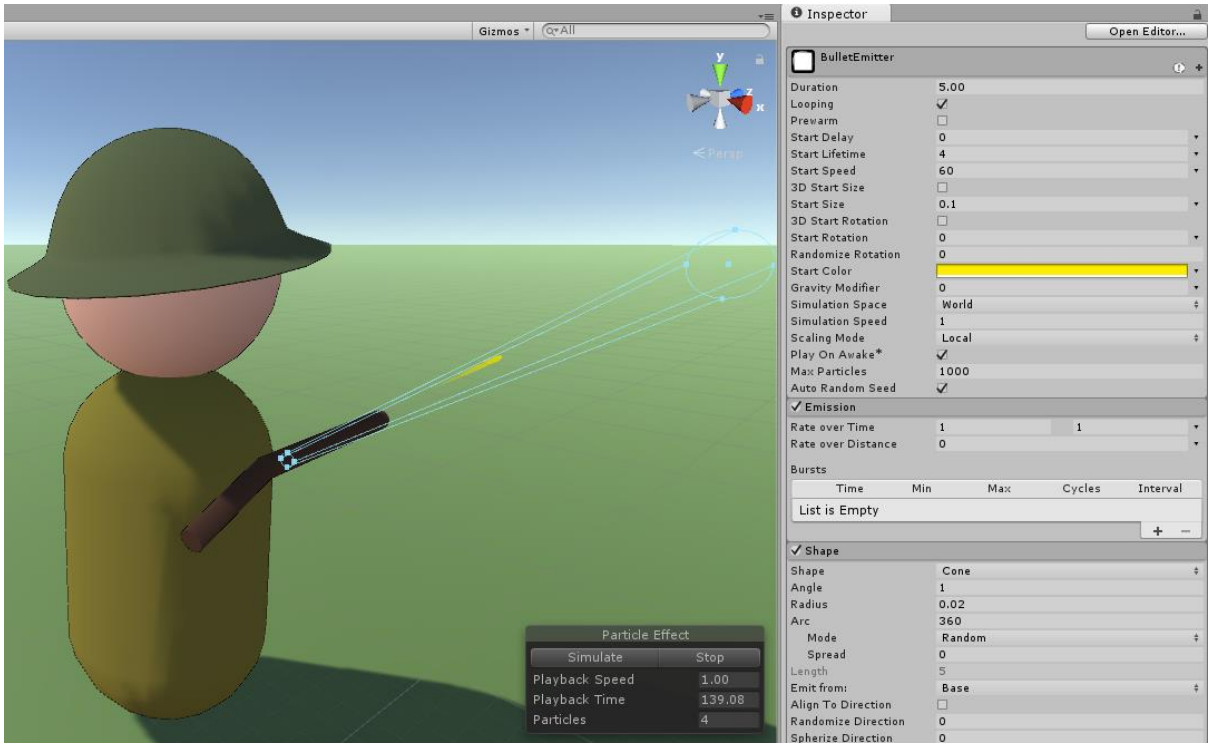


Figure 13. Creation of bullet emitter particle system for the infantry rifle

For the sniper, the start speed was simply made much faster, the trail made longer, and the radius/angle of the cone determining accuracy made smaller.

A script was made to be attached to each bullet particle system in which the *OnParticleCollision()* function, automatically called whenever a particle hits an object, determines whether it has hit a soldier and, if so, sends a message to its script, passing to it the specific collider hit. A function in the hit unit's script then determines how much health should be lost as a result. If the body is hit, 5 health is lost (half health of infantry or grenadier, full health of sniper or heavy weapon crew). If the head is hit, 10 health is lost (will always lead to death). If the helmet/hat is hit, no health is lost, but it becomes a separate game object and rigidbody and flies of the soldier's head, leaving it more exposed.

After the infantry and sniper were complete, work began on the tanks, since the functions of units like the grenadier and AT gun wouldn't be able to be tested until tanks were in the game. It was important to make their movement look believable, which was achieved with a

function to rotate smoothly and by limiting the tank's forward/backward acceleration based on how far from its target it's pointing. Forward and backward movement is done with an *AddForce()* function on the rigidbody, and experimentation and adjustments needed to be made to its mass and drag values along with the strength of the force until it looked and worked as envisioned.

The turret rotates smoothly in the same way as the hull, and the gun rotates around its pivot linearly. The functionality of the tank gun was implemented similarly to the bullets. It was considered to use game objects and rigidbodies for the shells, as it was thought they would look more realistic when bouncing off targets and the fact there would be fewer shells than bullets meant that performance wouldn't have been an issue. But there would have still been issues with detecting collisions accurately, and it was found that the desired effects could be largely achieved with a particle system.

A basic 3D mesh was made for a tank shell and used in the particle system's renderer without shadows to give the particles the shape of a shell, while still looking almost 2D and cartoon-like.

As with the bullet emitters, there is a script for the shell emitter. As well as handling collisions, it also ensures that the shell always points in the right direction when it is 'fired' by setting its start rotation to be the same as that of the particle system itself, which moves and rotates in line with the gun.

Figure 14. A Panzer IV tank in-game firing a shell

Upon a collision, the script determines first what the shell has hit. If it's anything other than a tank, such as the ground or a soldier, then it destroys the particle and triggers an explosion, consisting of two other particle systems which are children of the shell emitter

set to the world position of the collision. A function in the *Unit* class is called which has an explosion force added (and where applicable, health taken away) to all objects within the explosion radius. If a tank is hit, then its *penetrated()* Boolean function is called. This returns whether the shell would penetrate based on the shell's penetration value (defined by public variable on shell script) modified by the distance it's travelled and the effective thickness of the armour. This effective thickness (T) is calculated with the thickness of the armour represented by the particular collider hit (h) and the angle of impact (α) using the formula $T = h/\cos(\alpha)$.

h – actual armour thickness
α – impact angle (from normal)
T – effective armour thickness

$$T = \frac{h}{\cos(\alpha)}$$

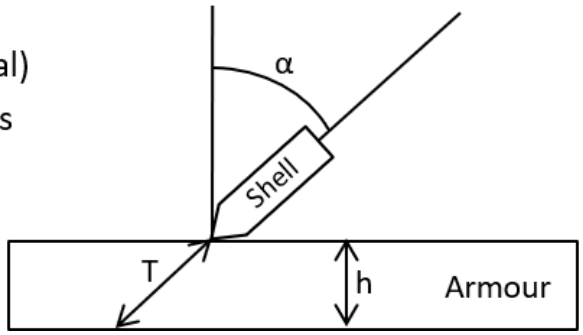


Figure 15. Diagram showing the way the effective thickness of armour is calculated based on the angle of impact

If the shell is determined to have penetrated the armour, then health is taken away from the tank (5 out of 10 max health) and the particle is destroyed, and a small 'puff of smoke' particle effect plays at the point of the penetration. A small force is also applied the tank in the direction the shell's travel. These effects give the player good feedback, making it both clear and satisfying when a tank is damaged. If a tank has run out of health, a plume of smoke with start rising from its rear, along with some sparks, making it clear which tanks are no longer in play.

If the shell doesn't penetrate, then the particle bounces off, and its rotation over speed is enabled, so it appears to spin as it ricochets off, slowing down if it comes to a stop on the ground.

The tanks' secondary weapon is the frontal hull machine gun. It was implemented using the same bullet emitter particle system as the infantry rifles (including the same script), but with a much higher rate of fire and much wider cone within which the bullets can deviate (lower accuracy). Each frame, the tank checks whether there is an enemy soldier close and within 30 degrees to the left or right of the direction the machine gun is facing and toggles it on if so. It acts independently of the turret, which can be shooting at a vehicle while the machine gun is firing at something else. For balance reasons, its range is smaller than the distance from enemies at which the tank will stop, so its role is largely limited to defending against soldiers moving towards it.

After tanks, work began on the grenadier unit. Like the infantry and sniper units, it inherits from the *soldier* class, but has a lot more of its own code as well, starting with the function for throwing its grenade. Each grenadier has one which it throws when there is an enemy soldier or heavy weapon a certain distance in front of it, which it detects in the same way as the tanks' hull machine guns do, independently of its primary target. The grenade is separated from the unit transform and given a rigidbody, to which a force forwards and upwards is applied. The same script then counts down from its timer of 3 seconds (± a slight random variation) until the explosion is triggered. It uses the same particle effects and functions for applying damage and explosion force as those for the tanks.

The grenadier's primary function of using a handheld AT weapon to attack tanks required more new work. To be effective, they would have to flank around to the side and rear of tanks where the armour is thinner. All other units at this point simply looked at and moved directly towards their target. The solution was to create three new empty child objects for each tank; one a distance to its left, one a distance to its right, and one a distance to its rear. When targeting a tank, a function in the *Unit* class determines which of these new 'marker' objects is the closest (the rear marker is defined as closest if the attacking unit is anywhere further back than the side markers). With the grenadier, the body rotates independently of the actual parent object rotation when attacking a tank, so it is always looking at it, even when moving towards one of the markers. The result is that they appear to strafe around tanks to the side and rear and fire at them.

The anti-tank weapons use the same shell emitter particle system as the tanks themselves, with the main difference being that there is no explosion when they hit something other than a tank. They will, however, knock back and kill any soldiers they hit directly.

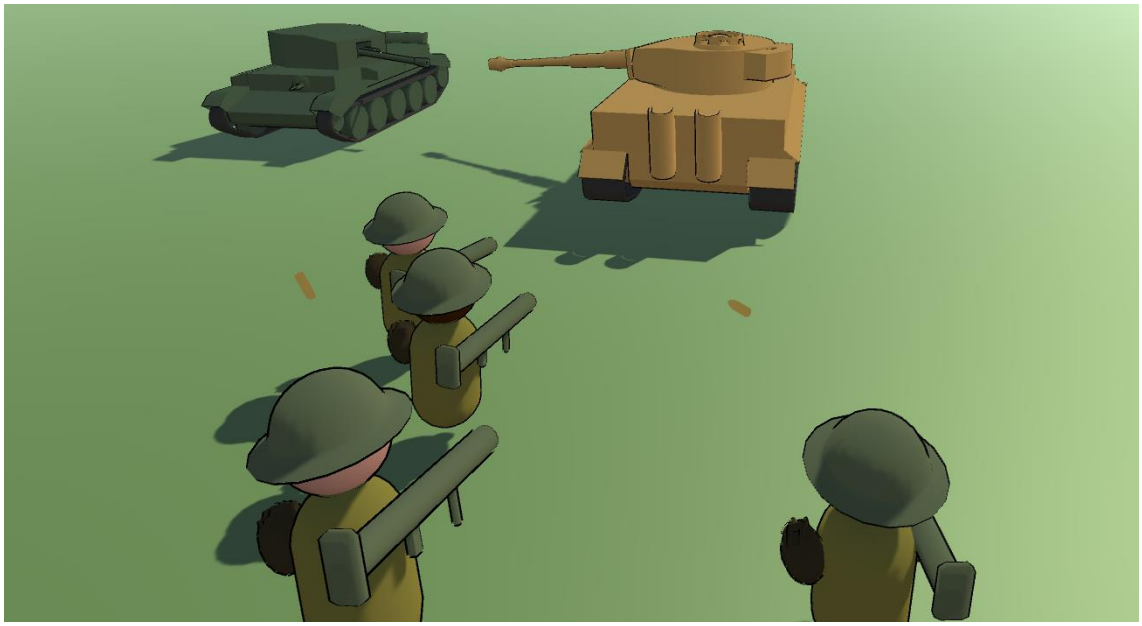


Figure 16. In-game screenshot of British grenadiers and a Cromwell medium tank flanking a Tiger heavy tank

Flanking was then implemented for medium tanks attacking heavy tanks, using the same marker child objects and functions for targeting them. The turrets already rotate independently of the hull, a very satisfactory functionality was achieved relatively quickly.

The final units to be programmed were the heavy weapons; the machine gun and anti-tank gun. Despite appearing quite different, they both inherit most of their behaviours from the *heavyWeapon* class. They each have to 'lift' their gun around a pivot when they move and put it back down once they're in range in order to fire. They have a limited arc within which the barrel of the gun can aim before they need lifting up and moving. The code works perfectly well for each with only different placements of the pivots and values for the rotation limits.

Their main difference is in their attacks. The anti-tank gun's shell emitter works in exactly the same way as the tanks', launching a single shell particle when the reload timer has reached 0 and the enemy is within range and line of sight. It has the same penetration value (130) and rate of fire as the heavy tank. The machine gun has a bullet emitter the same as those of the tanks' hull machine guns, but with a higher rate of emission, that is toggled on and off.

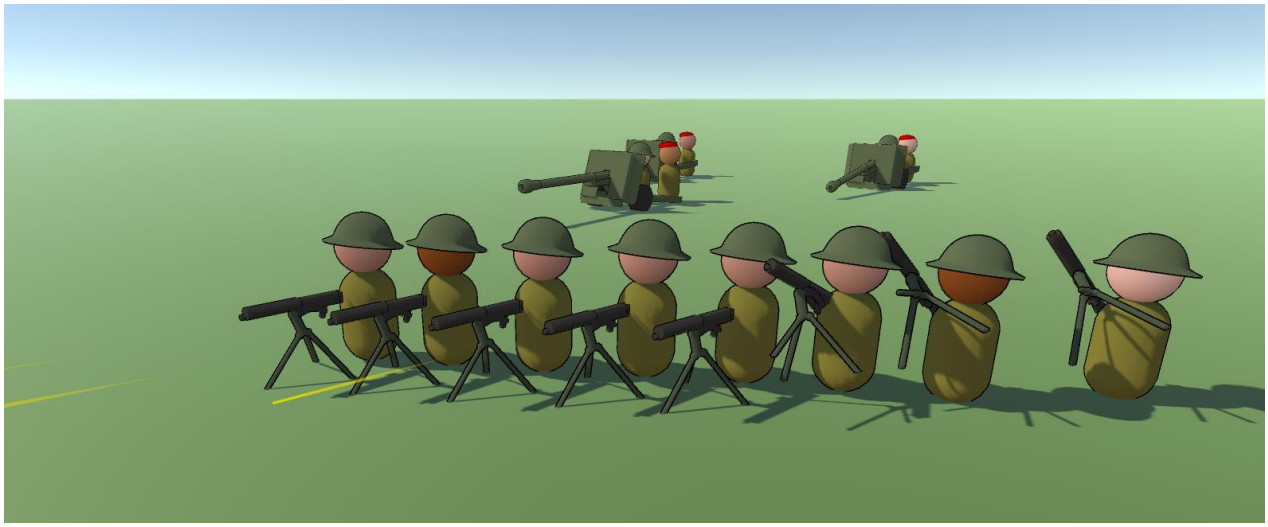


Figure 17. AT guns and machine gunners getting into a position to fire

One issue specific to the AT gun that needed solving came about from the fact it has two crew members. Initially, enemy infantry units would move towards and attempt to attack the shield of the actual gun as that's where the centre of the parent game object is. The solution was to use the marker points function used for flanking. AT guns now have side and rear empty child game objects just like the tanks, except the rear marker takes the position of the further-back crew member, moving to the other when that one is dead. Using the same function in the *Unit* class for detecting and moving towards to closest marker as grenadiers and tanks results in infantry going around to the sides and then attacking each

crewman. Each has its own health, able to take one shot each, and when they're both dead, the unit itself counts as being dead and can no longer move or attack.

Once all the units had been completed, the project needed to be turned into a functional game. Players needed the ability to choose and place units within a build, whilst testing so far had been done by simply placing prefabs into a scene in the Unity editor.

It would all be handled by the script attached to the camera that also handles its movement and rotation. The W, S, A and D keys move it forwards, backwards, left and right along a flat plane, the Q and R keys move it directly down and up, and moving the mouse changes the direction it's looking. The speed it moves depends on its height above the battlefield, moving more slowly when looking closely at units. It can be slowed down further by holding down the left shift key. Since watching the combat is all the player can do after placing their units, it was important to ensure that they have flexibility in how they watch it and can do so intuitively.

Units are placed within a grid, with each type taking a up a certain amount of space. A 2D array of transforms stores which squares are taken up by which units, and units cannot be placed or moved to a position where there is no space for them.

Two modes for placing units were included, which can be changed between using tickboxes in the editor UI canvas. Using move mode allows the player to click and drag to place a single unit in a specific location or move a unit that's already been placed. Clicking and dragging with draw mode on places a unit in every free space the mouse goes over, allowing large armies to be created quickly. Placed units can be deleted either by left-clicking them with the delete mode turned on, or by right-clicking them with any mode selected.

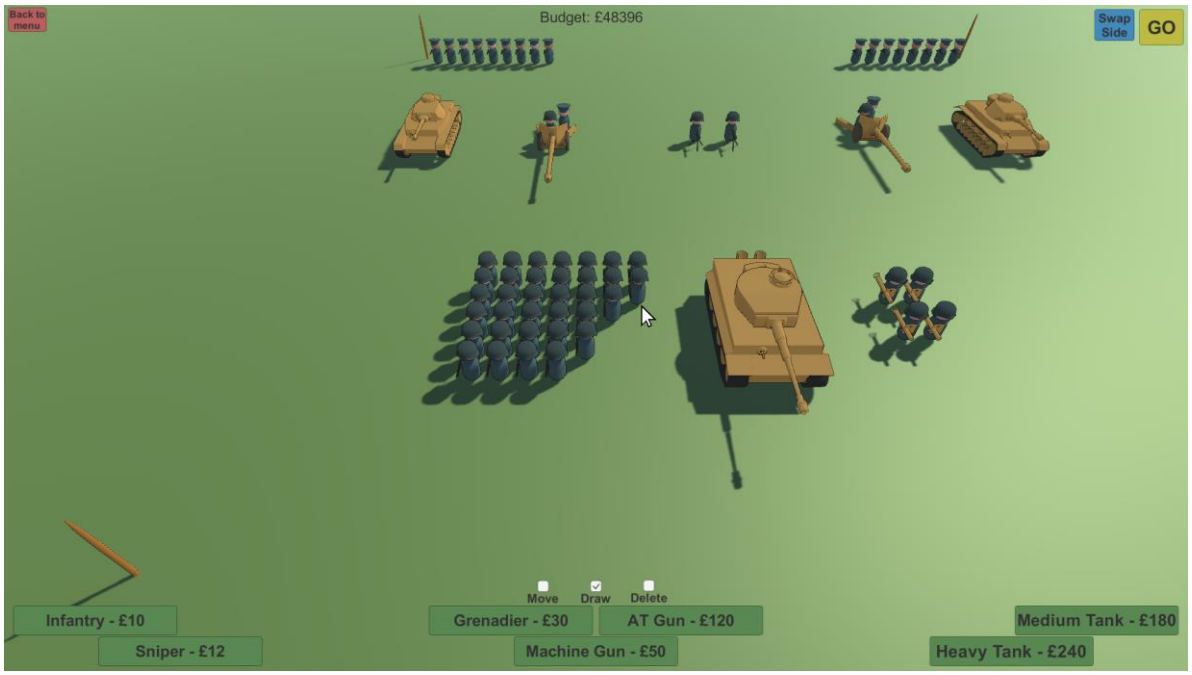


Figure 18. Screenshot of units being placed on the German side in sandbox mode.

The grid dimensions for each level are defined through public variables on the camera script. The boundaries are marked based on them using four marker objects, that look like large wooden stakes, at the corners. Also defined through public variables is the budget. It can be set differently for each level. Each time a unit is placed, the corresponding amount (returned by the *unitCost() function)* is subtracted from the remaining budget, shown at the top of the screen. The same amount is added back if a unit is deleted. If the player goes over budget, the overspend is shown in red and they are unable to press start.

If the player has placed an army that doesn't exceed the budget, they may press the 'Start' button in the top right of the screen to begin the fight. Doing so hides the UI for the editor mode, activates the scripts for all units and gives the player unrestricted movement with the camera (its rotation is fixed while placing units).

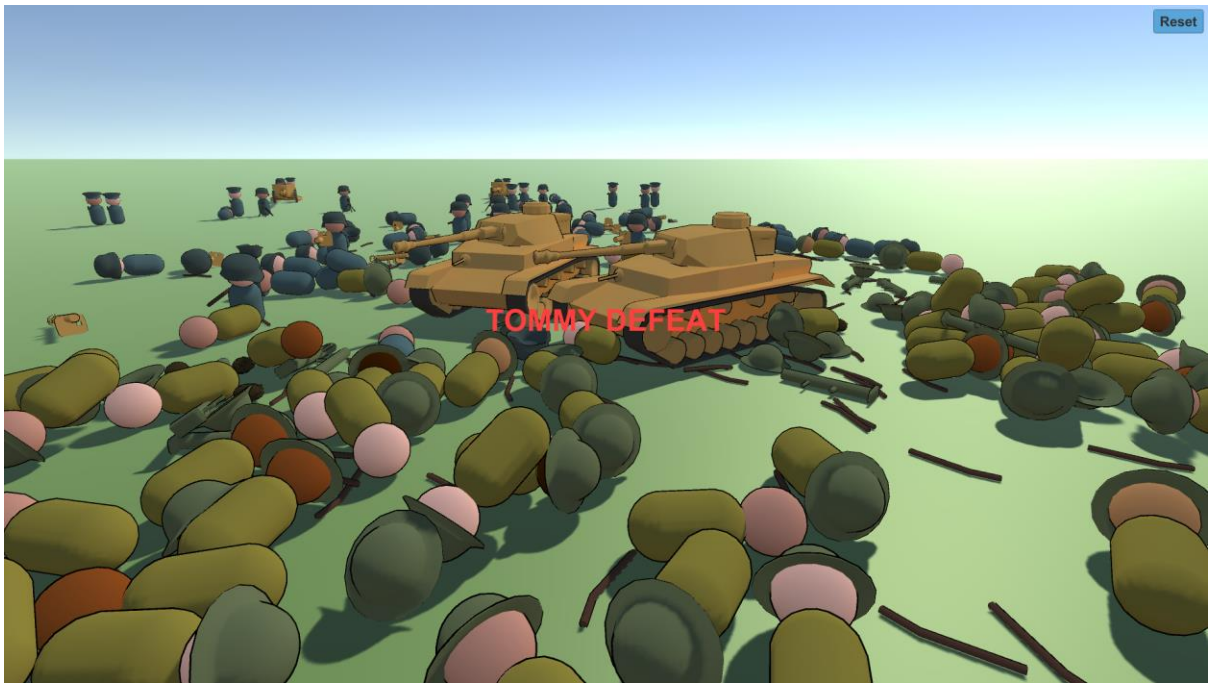


Figure 19. Screenshot of an attempt at a level ending in defeat for the player

The battle plays out, and once all units of one of the teams are killed, the result is displayed on screen. If the player succeeds in one of the levels, a button will appear to take them to the next level. If they are unsuccessful, they can press the reset button in the top-right of the screen, which is available as soon as the battle starts and resets the scene, so they can try making a new army.

Once all of the mechanics were completed, work began on the levels. They were made by running the game in the editor and placing a formation of German units, which were copied and then pasted into the level's scene and saved. The 'Swap side' button used in sandbox mode had to be enabled in order to make each level, then disabled again before saving.

There is a total of 12 levels. In the earlier levels, there are fewer units available to the player, with only infantry allowed in level 1 so it functions as a basic tutorial. Which units are available or locked is decided by a public string variable in the camera script. For example, the value “uptoatgun” will disable the ‘Button (Script)’ component of the medium tank and heavy tank buttons and replace their text with “Locked”.

In the later levels, more units are available to the player and budgets are higher, and they will be against larger numbers of German units.

Once the levels were completed, in order to make the game playable for testers, only the menu/user interface was needed to be completed to allow navigation between them. A scene was made for the main menu, which would be the first one seen when the game is run. It contains a UI canvas with buttons for each of the levels, for sandbox mode and to quit the game. They each link to a small C# script called *MenuScript* with two functions; one to load a particular scene based on the passed string for scene name, and one which simply quits the application.

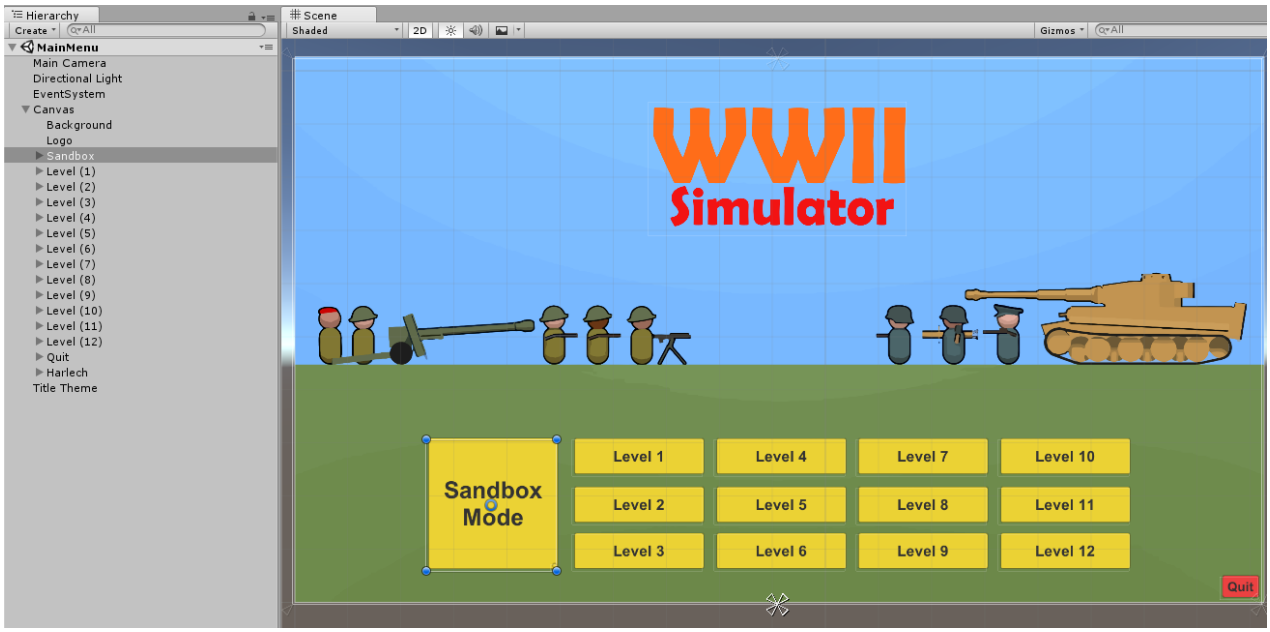


Figure 20. The Main Menu canvas shown in the Unity editor

A background image and title/were added to make the screen visually appealing, as it’s the first thing a player will see when playing the game. The background image was made by taking a screenshot of a build of a scene with an orthographic camera looking at an arrangement of static British and German units. The logo was made using text in Paint.NET (dotPDN LLC 2016).

3.4 AUDIO

Work began on implementing sounds began once all other aspects of the game were complete and functional, as they were less of a priority. But it was believed that they were an important feature that could only improve the game, and early testing responses showed that it was something players desired.

Royalty-free sound effects for a gunshot (Ruok 2015), explosion (Blastwave FX 2015) and a metal impact (Zapsplat 2015) were downloaded from online library *Zapsplat*. The audio-editing software Audacity (The Audacity Team 2016), was then used to alter their pitch and duration to create specific sound effects for rifle fire, tank gun fire, AT weapon fire, grenade and tank shell explosions and for the ricochet of a shell off a tank. For music, a medley of 'Songs of the World Wars' (Band of the Royal Corps of Engineers et al. 2002) including *Hang out the washing on the Siegfried Line* was bought, downloaded and edited to be faster and higher-pitch, to play in the Main Menu scene.

To be implemented into *Unity*, audio source components had to be added to the objects from which they would be heard, and the line of code to make them play added to their respective scripts. The curve for the volume per distance for each audio source component was adjusted so, for example, louder sounds like explosions and sniper rifle shots could be heard from further away than others.

4. TESTING AND REFINEMENTS

4.1 TESTING METHOD

Volunteers were sought to take part in the testing phase, asked either through word of mouth or online. Each was sent a link to a shared Google Drive folder. The first of its contents was a pdf 'Tester Guide' (Appendix B), which explained how to download and play the game, and the aims of the testing. It asked testers to first complete a short form (Appendix C), also in the Drive folder, which asked first for some basic information of them (their age, gender and computer gaming experience). They then had to confirm that they had had the testing explained to them and consented to take part by ticking tickboxes and signing their name.

The main aim of the testing, as explained to testers, along with getting general opinions on the game and finding any bugs, was to ensure that the units were well balanced. Specifically, that no units were too weak, too powerful, too cheap or too expensive, and that there were no 'one-size-fits-all' strategies. Players were advised to play the levels, testing different combinations of units, as well as experimenting in sandbox mode.

In order for participants to submit their feedback, a document was created for each of them (Appendix D). They contained questions on their opinions on the game, whether they found any bugs or other irregularities and whether they felt there were any balance issues.

4.2 RESULTS AND CHANGES

A total of 11 completed feedback forms were returned by test participants. In terms of general views of the game overall, all expressed a positive opinion of it. Several talked about how much they enjoyed the challenge of completing all of the levels. There was a lot of praise for the graphics; the cartoon art style was popular, along with specific details like the way helmets pop off or soldiers are sent flying by explosions. They really enjoyed watching the battles play out. Many also complimented the simple layout of the GUI, which they found easy to use.

There were relatively few balance issues reported, with some unable to suggest any unit which they thought was over- or under-powered, which is positive. Many appeared to pick up the dynamic of certain units countering others and how to use it to their fairly quickly. But there was a consensus amongst several that AT guns were too weak, particularly against medium tanks. One tester described needing four AT guns to be able to defeat just two medium tanks, which, as well as being unrealistic, shows that their strengths were not sufficiently counter-balanced.

As a result of this finding, both units were rebalanced in a number of ways. For the AT gun, the rotation speed, accuracy, maximum firing distance and rate of fire were all increased slightly. Their ability to aim at moving targets was also improved. The first change made to the medium tank was to make the colliders for the treads count as normal armour. Previously, any shell which hit them instead of one of the main hull and turret armour colliders would automatically ricochet. This led to many shells from AT gun shells bouncing unexpectedly off the sides of what were meant to be lightly-armoured tanks. This change made games with medium tanks more consistent and less frustrating, as there is now less luck involved in whether they get penetrated by a high-powered gun. The rate of fire was also reduced slightly.

Some testers gave feedback about the difficulty of the levels. This was used to ensure that none is too easy or hard, and that they increase steadily in difficulty. For example, Level 6, which had been judged to be significantly easier than the levels both before and after it, had the number of infantry supporting the enemy tanks greatly increased.

A few bugs and other issues were reported. While none were game-breaking, most of those which did have an impact were fixed either while the test was ongoing or afterwards. Several of these related to units either getting stuck or not firing when there were lots of destroyed tanks or heavy weapons on the battlefield. Existing code for flanking and strafing were used to solve many of the issues of getting stuck, and heavy weapons were changed such that dead units in front wouldn't stop them firing. One glitch which wasn't solved was one whereby a machine gunner unit would jump to the middle of the battlefield and spin. This appears to be purely visual, however, and doesn't affect the gameplay

Many testers suggested additions or changes which they thought would improve the game. As mentioned in chapter 3, there were those who would have liked sound effects and menu music, which was added shortly after. Other suggestions were for small improvements like the ability to see the enemy army before pressing go, for units to automatically stop being placed when the budget's run out and for draw mode to be the default mode for placing units, as it was most people's first choice. There was also an idea for a version of sandbox mode in which an enemy army would be procedurally generated based on how much the player has spent on theirs.

5. CONCLUSIONS AND FUTURE WORK

5.1 CONCLUSIONS

The key project task of designing, implementing and testing a complete, functioning real-time strategy game has been completed successfully with *WWII Simulator*. The results of the testing show that the game also achieves what it set out to do. Participants responded well to the game and spoke of their pleasure at the concept and art style and enjoyment of figuring out the strategies in order to overcome the challenge of the levels. This vindicates the design decisions and the literature that informed them as well as the implementation itself. On the main focus of balance of the units, the fact the few issues raised by the testing have all been addressed arguably serves as proof that ambition of making the game well-balances has been achieved. The value of the testing phase has also been clearly demonstrated.

The hypothesis that a large number of elements with simple rules in a video game could lead to what could be described as an example of emergence which serves as an important source of enjoyment for the player. The behaviours of the individual units in *WWII Simulator* are very simple, consisting largely of the basic functions of moving in the direction of the closest enemy and firing their weapon. The battles that play out when they're multiplied tens or even hundreds of times give the impression of greater complexity which player enjoy watching, as shown by the testing feedback.

5.2 FUTURE WORK

Ambitions for future further development of *WWII Simulator* would include suggestions made by participants of the test and proposed content and features which weren't included in the final version for this project.

There had been basic plans to include plane units such as the Spitfire fighters. The advice of Schell and Rogers in their books contributed to the decision to drop them. Without a lot more work, it was thought that they would either have very little relation to what happened to the other units, or would simply disrupt their balanced dynamics, without added much value to the project. Similarly, an ambition to include US and Soviet armies wasn't fulfilled as the lack of meaningful addition to the core gameplay wouldn't have justified amount of time required. However, they could be a successful part of any future expansions to the game separate from this project.

The small changes suggested by testers such as the ability to see the enemy army would be implemented, as would the ability to save army formations that players have made. The suggested feature of procedurally-generated enemy armies based on the player's would also be fascinating to develop, as it could add a lot of longevity to the game without simply creating more set levels.

5.3 FINAL WORD

This project's stated task of 'Design, Implementation and Testing of a Real-Time Strategy Game' has been successful, with all of its key aims achieved, including the key aim of ensuring it is well balanced.

REFERENCES

Schell, J., 2014. *The Art of Game Design.* 2nd edition. Burlington, VT, United States: Taylor & Francis Inc.

Portnow, J and Floyd, D. 2012. Perfect Imbalance - *Why Unbalanced Design Creates Balanced Play - Extra Credits* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=e31OSVZF77w [accessed 15th May 2018].

Rogers, S., 2010. *Level Up!: The Guide to Great Video Game Design.* 1st Edition. Hoboken, NJ, United States: John Wiley & Sons Inc.

Relic Entertainment, 2006. *Company of Heroes* [computer game]. Agoura Hills, CA, United States: THQ.

Plunckett, C., 2016. The Best World War Two Video Games. *Kotaku* [online]. 2nd February 2016. Available from: https://thebests.kotaku.com/the-best-world-war-two-video-games-1755330908 [accessed 14th May 2018].

Landfall Games, 2016. *Totally Accurate Battle Simulator.* Pre-alpha [computer game]. Stockholm, Sweden: Landfall Games.

The Family Gaming Team, 2017. *EATING PEOPLE!! Totally Accurate Battle Simulator #1! 3 Million Subs (FGTEEV TABS Gameplay / Skit)* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=tTnhGE1ZZ1I [accessed 14th May 2018].

Burbeck, S., 2004. *Complexity and the Evolution of Computing: Biological Principles for Managing Evolving Systems* [online]. Available from http://evolutionofcomputing.org/Complexity%20and%20Evolution%20of%20Computing%20v2.pdf [accessed 16th May 2018].

Gardner, M., 1970. The fantastic combinations of John Conway's new solitaire game "life". *Scientific American*, (Issue 223), 120-123.

Conway, J, 2014. Interview. *Inventing Game of Life – Numberphile* [video, online]. YouTube: Numberphile. Available from: https://www.youtube.com/watch?v=R9Plq-D1gEk [accessed 15th May 2018].

Ascani, E, 2011. *epic conway's game of life* [video, online]. YouTube. Available from: https://www.youtube.com/watch?v=C2vgICfQawE [accessed 15th May 2018].

Chan, C., 2016. This Totally Accurate Battle Simulator Might Be The Most Ridiculous Video Game Ever. *Gizmodo* [online], 14th July 2016. Available from: https://www.gizmodo.com.au/2016/07/this-totally-accurate-battle-simulator-might-be-the-most-ridiculous-video-game-ever/ [Accessed 28th October 2017].

Duffy, Q., 2016. The making of Company of Heroes: prototypes, design and the 'Donkeyschreck'. Interview with Tom Senior for *PC Gamer* [online], 13th October 2016. Available from:

http://www.pcgamer.com/the-making-of-company-of-heroes-design-prototypes-and-the-donkeyschreck/ [Accessed 28th October 2017].

Pegler, M., 1996. *British Tommy 1914-1918.* 1st edition. London, United Kingdom: Osprey.

Bowman, C and Mollow, S., 2017. *M5158475 - German boy soldier POW jeep* [photograph]. Harrowgate, United Kingdom: The Inter Group. Available from: https://www.flickr.com/photos/britishjeep/34626729331/in/photostream/ [accessed 17th May 2018].

Greene, J., 2017. 5 Reasons You Need a Game Design Document. *Luminosity,* 12th July 2017. Available from: http://luminositymobile.com/5-reasons-need-game-design-document/ [Accessed 3rd May 2018].

Autodesk, 2015. *Maya 2016.* [computer program]. San Rafael, CA, United States: Autodesk.

Pixologic, 2015. *Zbrush 4R7.* [computer program]. Los Angeles, United States: Pixologic.

EVOlution Graphics, 2009. *The-blueprints.com.* [website]. Hilversum, The Netherlands: EVOlution Graphics. Available from: https://www.the-blueprints.com/ [Accessed 15th November 2017].

Unity Technologies, 2015. *Unity.* 5.6.1f1 Personal. [computer program]. San Francisco, United States: Unity Technologies.

Microsoft, 2015. *Visual Studio 2015.* [computer program]. Redmond, WA, United States: Microsoft.

Warhead-Designz, 2015. *UniTOON Ultra* [download]. Savannah, GA, United States: Warhead-Designz. Available from: https://assetstore.unity.com/packages/vfx/shaders/unitoon-ultra-26776 [Accessed 4th December 2017].

dotPDN LLC, 2016. *Paint.NET.* 4.0.12. [computer program]. San Francisco, United States: dotPDN LLC.

Ruok, J., 2015. *Gun, Tikka T3 Battue 308 hunting rifle, gunshot, strong 4* [download]. Mudjimba, QLD, Australia: Zapsplat. Available from: https://www.zapsplat.com/music/gun-tikka-t3-battue-308-hunting-rifle-gunshot-strong-4 [accessed 7th May 2018].

Blastwave FX, 2015. *Grenade explosion, debris, blast* [download]. Mudjimba, QLD, Australia: Zapsplat. Available from: https://www.zapsplat.com/music/medium-large-metal-impact-wobble-2/ [accessed 7th May 2018].

Zapsplat, 2015. *Medium large metal impact, wobble 2* [download]. Mudjimba, QLD, Australia: Zapsplat. Available from: https://www.zapsplat.com/music/grenade-explosion-debris-blast/ [accessed 7th May 2018].

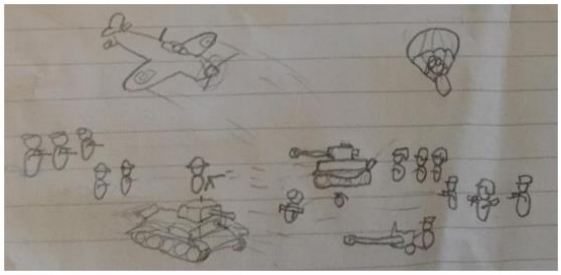
The Audacity Team, 2016. *Audacity.* 2.1.2. [computer program]. United States: The Audacity Team.

The Band of the Royal Corps of Engineers et al., 2002. *Songs of the World Wars* [download]. London, United Kingdom: Knellar Hall. Available from: https://itunes.apple.com/fr/album/kneller-hall-live-vol-3-golden-jubilee-concert/332012585?l=en [accessed 7th May 2018].

APPENDICES

APPENDIX A – GAME DESIGN DOCUMENT

WWII Simulator (Working Title)



Game Design Document

Written by Jake Ruggier

1.0 Overview

1.1 Theme / Setting / Genre

A light-hearted take on the real-time strategy genre in a World War II setting. It will feature soldiers, weapons and vehicles of different nations or 'teams' from mid to late WWII. These will be Team Tommy (United Kingdom) & Team Jerry (Germany). Long term, Team Ivan (Soviet Union) and Team Yank (United States) may be included.

A big part of the light-hearted, slightly comical theme will be the art style. Soldiers will be represented by a simple capsule for a body with a plain white sphere with a recognisable hat or helmet as the head. All models will be made up of cel-shaded block colours.

The player will have to complete battles based loosely on actual battles/theatres of the war. They will take place on simple maps like largely empty fields or deserts. There will also be a sandbox mode.

1.2 Core Gameplay Mechanics Brief

WWII Sim is a simple real-time strategy game. For each level, the player will have a limited budget to spend on units to make up a force to take on a particular enemy one.

The units each behave differently and go from the cheapest, the standard infantry units, to units like snipers and machine guns to the most expensive tanks and planes.

After choosing and arranging their units, the player presses play and watches their little army fight the enemy's army. If your little men manage to kill every enemy unit, then you win. But if they all get killed first, you can immediately try a different combination and formation of units, and try the battle again.

Players will need to think about whether they have the right balance of units to counter the enemy's, for example whether they have enough tanks to counter machine guns, and enough anti-tank guns to counter tanks.

There will also be a sandbox mode, in which players can set their own budget and choose the units for both teams, to see them duke it out. This could also function as a local two-player mode.

1.3 Targeted Platforms

The game will be built in Unity for release on PC through the Steam platform.

1.4 Project Scope

1.4.1 Game Time Scale

The game will be expected to take around 5-8 to complete. The assets and gameplay mechanics are expected to be largely implemented at least a couple of months before it's finished. But in a game like this, balance of units and costs is extremely important, and a lot of time will need to be spent testing and fine-tuning, to ensure, for example, that everything functions as expected and different battles can't all be won with just a large number of the same one or two units.

1.4.2 Team

Jake Ruggier – Lead designer, 3D modeller and programmer: Jake will be the source of ideas and the maker of design decisions. He will work on design and mechanics documents for the game. He will be responsible for all 2D and 3D assets. He will implement the game fully in Unity including programming of all C# scripts.

1.5 Influences

Totally Accurate Battle Simulator (Game):
Similar gameplay, the idea of a simple, comical game in which you choose units and watch them fight the enemy units.

Company of Heroes (Game):
A very good, more realistic WWII real-time strategy game featuring armies of the UK, US, USSR and Germany.

World of Tanks (Game):
Source of interest in World War II tanks and knowledge of how their armour works.

WWII films, e.g. A Bridge too Far, Fury, etc.:

A number of the many films set in World War II are big influences on my (and potential audiences') interest in WWII combat, and the game aims to produce a very simplified simulation of that.

1.6 The Elevator Pitch

A simple, light-hearted World War II strategy game in which you choose units, from different soldiers to tanks and planes, and watch them fight the enemies.

1.7 Project Description

A simple, comical World War II strategy game. For each level, you have to spend a certain budget on different units, including standard infantry, snipers, machine guns, tanks and planes. Your units hail from the UK or Germany
You then press play and watch your little army fight the enemy's army. If your little men manage to kill every enemy, then you win. But if they all get killed first, you can just try a different combination and formation and try again.

2.0 What Sets This Project Apart?

- Simple, light-hearted take on a WWII game.
- Simplistic, cartoon-like stylised art style.
- The ability to choose a large number of units and immediately watch them battle.
- Both fun, casual sandbox and challenging strategy.
- Soldiers and vehicles from different nations represented.

3.0 Story and Gameplay

3.1 Story

There will be no actual story, but levels will be rough takes on non-specific World War II battles.

3.2 Gameplay

The player will control a free third-person camera around the battlefield. At the start of a level, the player will be shown a list of units that can be used, their costs and the player's total remaining budget. The player will click the chosen unit type, and click in the scene to place them. They will be able to switch between 'single' and 'draw' modes. In the single mode will place a single unit of the chosen type and move with cursor if the mouse button is held down. In the draw mode, the player will be able to drag the mouse to place as many units as their budget allows in one motion. All units will be snapped to a tight grid.

They can then drag-select units and place them wherever they want (within certain bounds) and in custom formations by dragging the mouse (this will work similarly to the pre-battle setup in the Total War games).

When the player presses the start battle button, all of their units (and the enemy units) will come to life and start moving towards their enemies. The player had no further input into the battle and will move around to watch it play out. If the player's units manage to defeat all of the enemy units, they win and the level if passed if they are playing the set campaign missions. If all the player's units are defeated, then the player will need to press the reset button (which can be pressed at any point during the battle) which will take them back to the unit placement stage. They will be able to try as many different combinations and arrangements of units as they wish until they win.

4.0 Assets Needed

4.1 2D

4.1.1 Textures

Using block colours for all models.

4.1.2 UI

Simple text and buttons.
Background image and logo for title screen.

4.2 3D

4.2.1 Characters List

People

- Tommy with helmet
- Tommy with beret
- Jerry with helmet
- Jerry with cap

Ground Vehicles

- Cromwell
- Churchill
- Panzer IV
- Tiger

4.2.2 Weapons

- Rifles
- Grenades
- Handheld AT-launchers
- Anti-tank guns
- Machine guns

4.3 Sound

4.3.1 Sound List

- Sounds of shells hitting tank armour, both bouncing and penetrating sounds.
- Rifle and machine gun gunshot sounds
- Explosions of grenades and tank shells.
- Sounds of tank gun and AT weapons firing.
- Music for the title screen/main menu

4.4 Code

4.4.1 Scripts

- Parent Unit script
- Soldier script
- Child Infantry script
- Child Sniper script
- Child Grenadier script
- HeavyWeapon script
- Child MachineGun script
- Child ATGun script
- Tank script
- Child MediumTank script
- Child HeavyTank script
- BulletEmitter script
- ShellEmitter script
- Camera script
- Navigation/UI script

4.5 Scenes

- Main menu scene
- 12 levels
- Sandbox mode

APPENDIX B – TESTER GUIDE

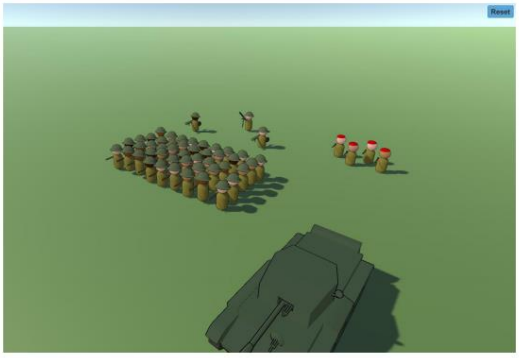
WWII Simulator Beta Test

Thanks for taking part! WWII Simulator is the game I have been making for my BSc Games Technology final year project, 'Design, Implementation and Testing of a Real-Time Strategy Game'.

It is a stylised, light-hearted take on World War II battles in the form of a simple and casual real-time strategy game. The objective for each level is to put together an army that can defeat the enemy army. You choose from a number of different types of units from Team Tommy (British) or Team Jerry (Germans), including basic infantry, snipers, machine guns and tanks, limited by a budget. Once you've placed your units, press go and watch the battle unfold. If they manage to defeat all of the enemies without getting wiped out themselves then you win, otherwise, just press the reset button to try a new combination.

How to play:

This is the screen you see when you start a level. To place units, first choose the type of unit you want to place by pressing its corresponding button, which also shows its cost, which will be taken away from your budget, shown at the top of the screen.
Use the tickboxes to choose a placement mode. Move mode allows you to click and drag to place a single unit in a specific location, or move around units you've already placed. Draw mode lets you click and drag to place a unit in every free space your mouse goes over, allowing you to make large groups of soldiers quickly. Right click units, or select the delete mode, to remove units from the scene and get the money back for them.
Once you've placed all of the units you want, and you haven't gone over budget, press the GO button in the top right to start the battle.



Your army will now start marching towards the enemy. You'll now be able to move the camera around freely (you can move the camera in edit mode, just less freely).

The controls are:

Move mouse – Look around.
W, A, S and D – Move forwards, left, backwards, right.
Q or left Ctrl – Move down.
E or Space – Move up.
Hold Shift – Slow down movement.
ESC – Pause.

If your army manages to kill every enemy unit without all dying themselves, you'll win and be able to proceed to the next level (though for this test, you'll be able to skip ahead to any level you want). If they don't manage it, you can press the reset button in the top right to go back to the edit screen to try a new tactic.

There is also sandbox mode, which can be accessed from the main menu. Here, you choose the units for both teams, and have an almost unlimited budget. You can play around here with different scenarios, like a massive line of snipers against infantry, or grenadiers only. Just bear in mind that too many soldiers at once will make the game run slowly, depending on how powerful your PC is. You can also use it for local 2-player. Each come up with your best arrangement of units, and see which comes out on top.

Unit guide

Infantry – Basic soldier unit with rifle.

Sniper – Can't take much damage, keeps distance and fires accurate shots, prioritising machine gunners and other snipers.

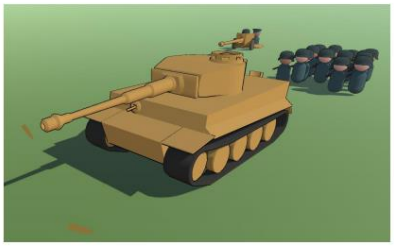
Grenadier – Tries to get around enemy tanks to fire their low-penetration handheld AT weapon into the thinner side and rear armour. Also has a grenade to throw.

Machine gun – Places itself at a distance from the enemy, very rapid but inaccurate fire.

Anti-tank gun – Keeps its distance and fires high-penetration shells at tanks, or other units if necessary, loses effectiveness at short range.

Medium tank – Fast moving, lightly armoured tank. Will attempt to flank enemy heavy tanks, also very effective against soldiers and emplacements.

Heavy tank – Slower tank with stronger frontal and side armour. Grenadiers and medium tanks will have a harder time penetrating it.

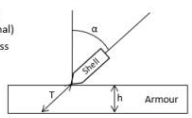


Tank shells – Each anti-tank gun's shells have a particular thickness of armour the can penetrate. Shells that are too weak or hit at too much of an angle will bounce off. Handheld AT weapons and medium tank guns can't expect to get through the front armour of a heavy tank, so must try and get around to the side and rear.

h – actual armour thickness
α – impact angle (from normal)
T – effective armour thickness

$$T = \frac{h}{\cos(\alpha)}$$

You don't really need to know this



Aim of the testing

While I do want your opinions of the game, that's not the primary aim of this beta test. Alongside ironing out any bugs, the main objective is to make the game balanced in terms of the units, i.e. none of the units are over-powered or obsolete. Although it is only a simple strategy game, balance is still very important, keeping the dynamic of the player choosing different units to counter specific enemies. For example, machine guns, which can decimate infantry, are easily destroyed by tanks, which themselves are balanced out by anti-tank guns and grenadiers. It's key that there is no 'one size fits all' strategy.

As well as simply completing the levels, try different combinations to see if there are any which allow you to win different levels without being forced to try different tactics. Experiment in sandbox mode as well. Take note of any units you feel are too powerful, too weak, too expensive or too cheap in relation to others.

Another aspect is the role random chance plays in each battle. Do you feel as if you win or lose battles based on the units you've chosen, or that it's too much down to luck? How much variation in results do you get from the same combination and placement of units?

Get started

Before you start playing the game, please complete the short pre-testing form on the Google Drive.

To play the game, download the Zip file from Drive. Once it's downloaded, extract it to a folder wherever you want it on your PC. You should then be able to run the WWIISim.exe application. If you can't get it running, or have any other problems, contact me on Facebook or at i7672007@bournemouth.ac.uk.

Once you've spent some time on the game, you can start filling in the feedback form in the Drive folder with your name on it (you can add to it as you play, doesn't have to be done all in one go). Any and all responses are useful, so don't worry if you haven't had that much time to spend on it.

Happy testing!

APPENDIX C – FORM TO BE COMPLETED BEFORE TESTING

Pre-Testing Form

Please complete this before you start.

*Required

Age

- ○ 18-24
- ○ 25-30
- ○ 31+

Gender

- ○ Female
- ○ Male
- ○ Other: _____

Gaming experience

- ○ Play games very little/novice
- ○ Sometimes play games/intermediate
- ○ Play games regularly/expert

*

- ☐ I confirm that I have had the testing explained to me/information provided.
- ☐ I consent to taking part in this test and for the data I provide to be used for the project.
- ☐ I understand any data collected will be anonymous and confidential, and used only for the project.
- ☐ I understand that I can withdraw my role in the testing and my data will be deleted if requested.

Name *

Your answer

SUBMIT

Name:

WWII Simulator Beta Test Feedback

Thanks again for testing! Once you've spent some time playing the game, please fill this in, ideally by Wednesday 9th May. You can fill it in all at once or bit by bit, and add to it as you play more or change your mind. Feel free to skip a section if you haven't got anything to say or don't have much time.

What were your first impressions of the game?

Any glitches/bugs/crashes/other oddities that you've noticed? (e.g. Particular unit won't fire, game crashes when _____.)

Balance issues: Any unit you feel is too powerful/weak or too expensive/cheap, any unit combinations that always work whatever they're up against, any level that can be too easy or too difficult. (e.g. Snipers too powerful compared to infantry, AT guns too weak, I can win most levels with this particular setup.)

What are your overall opinions of the game? Favourite aspects? Do you enjoy the gameplay/like the graphics? Do you like the challenge of the levels or prefer playing around in sandbox mode? How do think the game could be improved?

Bournemouth University

Research Ethics Checklist

Reference Id	18188
Status	Approved
Date Approved	09/01/2018

Researcher Details

Name	Jake Ruggier
Faculty	Faculty of Science & Technology
Status	Undergraduate (BA, BSc)
Course	BSc Games Technology
Have you received external funding to support this research project?	No

Project Details

Title	DESIGN, IMPLEMENTATION AND TESTING OF A REAL-TIME STRATEGY GAME
Proposed Start Date of Data Collection	05/03/2018
Proposed End Date of Project	18/05/2018
Supervisor	Jose Fonseca
Approver	Jose Fonseca

Summary - no more than 500 words (including detail on background methodology, sample, outcomes, etc.)

I plan to develop a complete 3D real-time strategy game for PC, a stylised take on World War II battles. This will include game design, 3D modelling, programming and Unity implementation, demonstrating my skills in these areas. The nature of strategy games like this means there will need to be an extensive testing phase in order to ensure that the game is balanced in terms of the different playable nations and unit type. Volunteers may be needed to help with this playtesting.

External Ethics Review

Does your research require external review through the NHS National Research Ethics Service (NRES) or through another external Ethics Committee?	No

Research Literature

Is your research solely literature based?	No

Human Participants

Will your research project involve interaction with human participants as primary sources of data (e.g. interview, observation, original survey)?	Yes
Does your research specifically involve participants who are considered vulnerable (i.e. children, those with cognitive impairment, those in unequal relationships—such as your own students, prison inmates, etc.)?	No
Does the study involve participants age 16 or over who are unable to give informed consent (i.e. people with learning disabilities)? NOTE: All research that falls under the auspices of the Mental Capacity Act 2005 must be reviewed by NHS NRES.	No
Will the study require the co-operation of a gatekeeper for initial access to the groups or individuals to be recruited? (i.e. students at school, members of self-help group, residents of Nursing home?)	No
Will it be necessary for participants to take part in your study without their knowledge and consent at the time (i.e. covert observation of people in non-public places)?	No
Will the study involve discussion of sensitive topics (i.e. sexual activity, drug use, criminal activity)?	No

Are drugs, placebos or other substances (i.e. food substances, vitamins) to be administered to the study participants or will the study involve invasive, intrusive or potentially harmful procedures of any kind?	No

Will tissue samples (including blood) be obtained from participants? Note: If the answer to this question is 'yes' you will need to be aware of obligations under the Human Tissue Act 2004.	No

Could your research induce psychological stress or anxiety, cause harm or have negative consequences	No

for the participant or researcher (beyond the risks encountered in normal life)?	
Will your research involve prolonged or repetitive testing?	No
Will the research involve the collection of audio materials?	No
Will your research involve the collection of photographic or video materials?	No
Will financial or other inducements (other than reasonable expenses and compensation for time) be offered to participants?	No

Please give a summary of the ethical issues and any action that will be taken to address these. Explain how you will obtain informed consent (and from whom) and how you will inform the participant about the research project (i.e. participant information sheet).
I don't believe any ethics issues arise from anything I might do. I will simply ask people to playtest my game and give feedback in through a form. Participants will be informed about the project the playtesting I want them to do, both orally and in writing, through a brief introductory document containing the relevant information, which will be given to them before they start. At the same time, a consent form will also be given to each participant to sign (he/she will keep one copy and the original will be kept by me).

Final Review

Will you have access to personal data that allows you to identify individuals OR access to confidential corporate or company data (that is not covered by confidentiality terms within an agreement or by a separate confidentiality agreement)?	No
Will your research involve experimentation on any of the following: animals, animal tissue, genetically modified organisms?	No
Will your research take place outside the UK (including any and all stages of research: collection, storage, analysis, etc.)?	No

Please use the below text box to highlight any other ethical concerns or risks that may arise during your research that have not been covered in this form.