A-Level Computer Science

“Metal Lynch”

Programming Project

George Osborne

Candidate Number: 8645

Hereford Sixth Form

Centre Number: 24175

Contents

[Analysis 3](#_Toc10488678)

[Problem Identification 3](#_Toc10488679)

[Thinking Abstractly 3](#_Toc10488680)

[Thinking Ahead 4](#_Toc10488681)

[Thinking Procedurally 5](#_Toc10488682)

[Thinking Logically 5](#_Toc10488683)

[Thinking Concurrently 5](#_Toc10488684)

[Stakeholders 6](#_Toc10488685)

[Research 6](#_Toc10488686)

[Initial Research 6](#_Toc10488687)

[Stakeholder Questionnaire 11](#_Toc10488688)

[Specification 11](#_Toc10488689)

[Hardware and Software Specification 11](#_Toc10488690)

# Analysis

## Problem Identification

I am going to be developing a 2d, multiplayer, turn-based strategy game where players play as tanks, that I am calling Metal Lynch. The aim of the game for each player is to reduce their opponent’s health to zero, and this may be accomplished by aiming their tanks cannon to shoot at the other player’s tank. It will take multiple hits to accomplish this, and so players are able to move their tank a limited distance each turn, in order to position themselves.

They will be able to select from a range of maps at the start at the game, and craters will be added dynamically to the map floor when a player’s projectile hits the floor. The player will also be able to pick from a range of projectiles to shoot at their enemy, each with varying properties such as damage and weight. This variety will help make the game more replay-able, as it won’t get boring and repetitive.

The game I am making is heavily inspired by a game called Shellshock Live, but will be different in some key ways (There will be a more in depth comparison between this game and the game I am making in the research section of the analysis). I plan on making my game a lot simpler, as it will not include items, and will have a smaller range of weapons and maps to choose from. This will help me make the GUI (Graphical User Interface) a lot simpler and more user friendly, which I am hoping will make the game more attractive to beginners, but also not oversimplify the core of the game.

I am also planning on creating a unique game aesthetic that contrasts greatly with Shellshock Live’s. Where Shellshock Live takes a more abstract design for the map, game objects and GUI, with heavy usage of vector graphics for the map and tanks, I am planning on giving it a more vivid design, where I will give the tanks and GUI a greater sense of realism. I am going to be using imagery of real, modern day tanks to inspire how I design the tanks and GUI, however I will not overcomplicate the design so that the core gameplay is not convoluted, and so that the game runs at a high enough framerate to be fluent (I am aiming for around 30 frames per second).

In order to ensure that the game I am making is entirely solvable through a computational approach, I will justify how I will make the game using the 5 computational methods that I think are relevant.

### Thinking Abstractly

Abstraction is where a real life system is studied, so as to remove details and attributes that are unimportant to the problem being solved. This leaves only the relevant parts of the system that can be used to solve the problem computationally. For my project I will be studying the physics system under which tanks and projectiles follow in real life, and abstracting from this system for my game.

The first major difference between Metal Lynch and real life is that real life is 3 dimensional, whereas Metal Lynch is 2 dimensional. This is an easy abstraction to make from real life, because from now on when studying how projectiles and tanks work in real life, all we have to do is only move these objects in the X and Y plane, allowing us to completely ignore the Z plane.

Another big abstraction that I will make is when it comes to the trajectories of projectiles. Again, we will be ignoring the 3rd dimension in real life in order to model trajectories in real life. In Metal Lynch there will be a total of 3 forces that may act upon the projectile; gravity, wind resistance, and the initial force of the tank cannon. These will all be constant forces acting upon the projectile, which is simplified from real life where some of these forces such as wind resistance would not be constant, and could change while the projectile is in mid trajectory.

The last major abstraction that I will make is when it comes to the craters that projectiles will leave in the map. In real life these craters will vary hugely depending on the size and force of the projectile, and will likely not be perfectly circular. I will be heavily simplifying this by assigning certain crater sizes and shapes to certain weapons. This means that the crater generation will not take into account the size and force of the projectile but will be predetermined by the weapon being used.

### Thinking Ahead

Thinking ahead is where the programmer plans their approach to the problem beforehand, so that their solution is as efficient as possible. Thinking ahead helps prevent the unhealthy practice of writing the code on the spot, with no real long-term idea of how the program will look from the start. This practice is bad because it encourages the programmer to write untidy, inefficient code, and because it makes bug-fixing much more difficult.

First, I will identify the inputs and outputs required in my solution. The inputs will be a keyboard and mouse, and I do not intend to implement controller support because the precise aiming required in my game requires a mouse. Players will move the tank with the WASD keys, as this is standard in most PC games, and use the mouse to aim and fire. These controls will be relatively easy to implement in C#, which allows for events to be set up that are triggered upon certain actions being performed (such as a key press or a mouse movement). The outputs will be comprised of the display and the audio output (headphones/speakers). The display will be the primary output, as it is required for the players to play the game. The audio output will be used for sound effects and the soundtrack, and so not strictly required to play the game.

Next, I will explain the importance of reusable program components, and how I plan to implement these. For a problem as complex as this one, it is vital that I make use of reusable program components to maintain efficiency and simplicity in my solution. Reusing components not only saves time but also makes the code easier to read and troubleshoot. If I am reusing a piece of code that I know works from previous testing, I can be sure that that piece of code is not the source of any bugs I may be experiencing. Implementing reusable program components is relatively simple in C#, because I am able to create functions and procedures for different objects that can be used multiple times on instances of those objects. For example, one procedure that I plan on creating will launch a projectile. It will be reused multiple times in the program and will take parameters such as the initial velocity of the projectile, allowing it to give varying results despite being the same piece of code being used each time.

### Thinking Procedurally

Thinking procedurally is where the problem is broken down into smaller sub-problems that can be solved individually to create a final solution. This approach is necessary when solving a problem as complex as the one I face, because it allows us to focus on small, manageable chunks of the problem. Trying to solve the entire problem head on would be extremely difficult, but smaller problems can be solved much more quickly and effectively.

From how I see it, my problem can be broken down into two main sub-problems, and each of these must be broken down into smaller sub-problems. These two sub-problems are the graphical user interface (GUI), and the actual gameplay itself. The GUI can be broken down into smaller sub-problems, which are the main menu screen, the settings screen, the game options screen, and the stage selection screen (these are the main ones, but I might add more later in development). The actual gameplay can be broken down into the game objects, which are the tanks, the projectiles, and the map, and the physics system.

### Thinking Logically

Thinking logically is where a problem is looked at as a series of logical decisions that can be made up into an algorithm. This is essential because it is fundamentally how computers work; computers perform tasks by making a number of logical decisions.

As demonstrated in the previous section (thinking procedurally) my problem can be broken down into many smaller sub-problems. These sub-problems can be looked at as algorithms, each containing a series of logical decisions. Trying to examine the entire problem as a huge algorithm would be incredibly difficult, but now that the problem has been broken down, it is easy to see how each sub-problem is just a simple algorithm. Going back to the previous example of the procedure to launch a projectile, it becomes easy to think about it logically. The first logical decision to be made is whether or not the launch button has been pressed. If it has, it can then start moving the projectile along its trajectory. While on this trajectory, it must continuously ask whether the projectile has hit an object. When it has, it can then decide whether or not the object was a tank or the floor, and either take damage off of the player or just add a crater to the position where it landed. By breaking down each sub-problem logically like this, it is possible to solve the entire problem algorithmically.

### Thinking Concurrently

Thinking concurrently is where parts of a problem can be solved at the same time. An example of how I will be thinking concurrently when approaching my problem is when I will be designing the different variations of the projectiles. When designing these variations, I can create their basic trajectory methods all at once, because they will all be based off of the same basic formula. This is helpful because it means that I won’t need to implement each of these trajectory methods individually.

## Stakeholders

I have chosen three stakeholders who have an interest in my project. These are Jack Kench, Daniel Matthews and Sebastian Roffey. I have chosen these because they are good friends who enjoy local multiplayer gaming. They are familiar with the turn-based strategy game genre but are looking for something new that will give them more control over the game experience through modifiers.

My game is intended for teenagers and older, primarily male, who are looking for casual fun with friends. I chose my stakeholders because they are within the intended age group (both are 17), are of the male gender, and they enjoy playing local multiplayer strategy games, giving them enough experience to be able to give constructive criticism that is reflective of how they enjoy local multiplayer gaming.

They are interested in my project because they enjoy the game genre, but feel like there is a gap in the market. They are looking for a turn-based strategy game that is not overly complicated, making it easy to understand and play with very few instructions, but also not so simple that it gets boring quickly. Only one of my stakeholders is familiar with the game that has inspired my project (Daniel Matthews) but he feels that this game doesn’t allow for enough customisation of the game experience. He is looking for a game that has more modifiers that can be changed that effect the physics of the game: e.g. the ability to change the gravity that pulls down on projectiles, or the amount of health the player starts with. I intend to fill this hole in the market with my project.

I believe that they will be useful in helping me achieve the best possible solution because they have experience with similar games in the genre, and so will know roughly what to expect of a game in this genre. They are also interested in playing this with other friends who are inexperienced in this genre, and so will be looking to judge the ease of use and the simplicity of the design of this game. All of this means that they will be able to provide useful feedback that will help me achieve the best possible solution.

One final note is that one of my stakeholders is dyslexic (Sebastian Roffey) and requires that I add different colour options to the game to increase the ease of use for him. This makes him an especially useful stakeholder because he will be able to provide feedback that will help me ensure that the game reaches its widest possible audience, and doesn’t drive away people who are dyslexic.

## Research

### Initial Research

As previously mentioned in the problem identification, my game is heavily inspired by a game called Shellshock Live. This game is available for purchase to download and play locally, but also has a similar, slightly simplified sequel available to play online as a flash game (<https://www.shellshocklive2.com/>). I will primarily be describing and comparing to this, the flash version of the game, because it is the one that my stakeholders are most familiar with.

In Shellshock Live 2, players must create an account because there is a system of online matchmaking, where players play against one another over the internet.



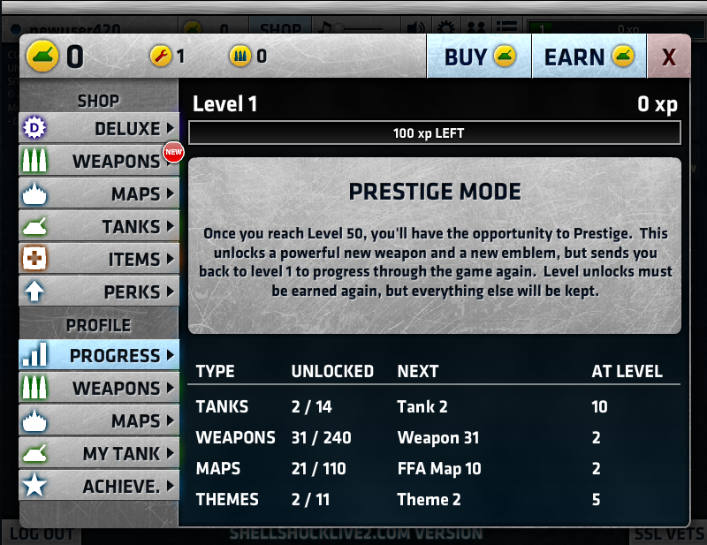
A look at Shellshock Live 2's online account system.

Once signed in, the main menu appears with a range of options.



Shellshock Live 2's main menu

Under the profile section the user’s progress is shown.



Shellshock Live 2's profile section showing the user's progress.

With this system, the more the user plays, and the more successful they are, the more tanks and weapons they unlock.

The button labelled ‘Firing Range’ brings the user to a tutorial/practice section of the game.



Shellshock Live 2's tutorial/practice level.

It is here that we get our first look at gameplay. As shown in the image above, the controls are the ‘A’ and ‘D’ keys to move the tank, and the arrow keys to aim/using the mouse to aim.

In this tutorial/practice level, the gameplay consists of practicing your aim on the target. The target does not move in this level, and its health doesn’t deplete either. The user can select from all of the weapons that they have unlocked so far or can change the map or wind level to their liking.

A picture containing electronics

Description automatically generated

Shellshock Live 2's shooting mechanic.

A screenshot of a computer

Description automatically generated

Shellshock Live 2's weapon selection.

A screen shot of a computer

Description automatically generated

Shellshock Live 2's map and wind selection.

After pressing the big ‘Play’ button, the following screen is shown, giving the user instructions on how to proceed.

A screenshot of a computer

Description automatically generated

Shellshock live 2 helpfully instructing the user how to join a game.

After joining a game, Shellshock Live 2 displays a matchmaking screen where players wait for enough players to join. While they wait they can chat, and once enough players have joined and pressed the ‘Ready’ button, the game starts.

A screenshot of a computer screen

Description automatically generated

Shellshock Live 2's matchmaking screen where players can chat as they wait for the game to start.

Once in a game, players take turns aiming and firing at one an over, with the goal of depleting the enemy team’s health to zero.

A screen shot of a computer

Description automatically generated

Shellshock Live 2 gameplay image 1.

A screen shot of a computer

Description automatically generated

Shellshock Live 2 gameplay image 2.

As shown in the two above images, as the game progresses and more shots are fired, craters will be added to the map dynamically where shots landed.

Once a team has one the game, a screen showing the player’s progress towards new unlockable rewards will appear.

A screenshot of a cell phone

Description automatically generated

Shellshock Live 2's victory screen, illustrating their progression system.

The game offers the following gameplay options to choose from: teams, free for all, 1-6 player battles, wind strength options, and time limits.

### Stakeholder Questionnaire

## Specification

### Hardware and Software Specification

To produce the game, I will be using Visual Studio 2017, which runs on Windows 7, 8.1, and 10. The minimum hardware requirements are the following: 1.8 GHz or faster processor (duel core or better recommended), 2 GB of RAM, video card that supports a minimum resolution of 720p (1280x720), 20 GB hard disk space.

The game will be distributed as an executable, which is a file that performs the encoded instructions without the need of a program for it to be parsed through. This means that users can simply run the executable in order to play the game, so long as they have a PC with Windows installed.