Warstrike

Team Project - Spring Term 2019

Final Report

Team Wipeout:
Anca Burduv
Andrei Mihai
Kelvin Chua
Liam Skop
Stefan Nedelcu
Weng Chan

March 2019

Contents

1 Introduction

Warstrike is a 2D side-view artillery turn-based tactics video-game. The player controls a tank in a battle against another human-controlled tank or computer-controlled tank depending on the type of game mode he/she chose to play. The player's and the opponent's tank are equipped with weapons based on real-life or fictional weaponry.

The game engine will link the user interface (UI) and the back-end system. The game engine has a game loop which will initialise the scene and it will have an update method which keeps track of the state of the game. It will render the

graphics at 60 frames per second. It also checks for user input at all times and will give a corresponding output if applicable.

2 Final Requirements

The initial stages of the development of the game involved using JavaFX libraries to render the graphics and the graphical user interface (GUI). The team found difficulties trying to integrate it with the game logic and believed that a simpler game engine is needed, hence Slick2D became the alternative option, as its simplicity allowed us to write the required game logic.

The team has also switched from Model View Controller (MVC) system architecture to a Entity Component Systems (ECS) system architecture. As our game is more data-oriented, and the team wanted flexibility in defining entities in the game where every object in the game (such as tanks and bullets). The reasons behind this transition will be elaborated further in the 'System Architecture' section.

Another major change was the transition from a turn-based multiplayer game to a real-time game. The team believed this change should be made, given the type of our game, this will make the matches and the combat mechanics more interesting for the players.

The team has also decided to introduce a more strategic gameplay experience for the players, therefore with the implementation of the existing fuel system which encourage players to move their tanks strategically, the team also fixed the initial velocity of the tank's bullet, so that players could estimate where the bullets would land and make more accurate predictions in combat. This made the idea of using a power slider that changes the initial velocity of the bullet futile hence this feature is dropped.

In order to cater to more players from different levels, the team has introduced three difficulty levels for the Artificial Intelligence (AI) player which are easy, medium and hard. This will be detailed in the "Artificial Intelligence" section of the report.

3 Software Design

3.1 Game Engine

Slick2D is a Java-based game engine. It is able to provide all the functionality needed for our game in the simplest form, removing all the complexity, making the game architecture clear to developers which allows us to understand the game inside out. Slick2D's simplicity allows implementation to take place quickly and produce a prototype in a short period of time which coincides with

our software engineering process model: incremental development, which will be further elaborated in the 'Software Engineering' section.

Slick2D is based on three fundamental methods which are `init()`, `update()` and `render()`.

```
public class MyGame extends BasicGame {
   public void init(GameContainer container) throws SlickException {
      ...
   }

   public void render(GameContainer container, Graphics g) {
      ...
   }

   public void update(GameContainer container, int delta) {
      ...
   }
}
```

The `init()` method is invoked at the start when the game is being executed. It is responsible for loading the necessary resources needed by the game such as images, sprites and audio files.

The `update()` method runs the game loop every time that game loop needs to be updated.

The `render()` method is invoked every time the game needs to be rendered. This renders the game state into a frame. The rendering frequency is dependent on the frame rate set, which in our case is 60.

These methods are defined in a class that inherits the `BasicGame` class, which is essentially the backbone of the game that includes the game loop and input handling.

3.1.1 System Architecture

The game uses the Entity Component Systems (ECS) software architecture. This type of architecture aims to create game entities by using the Composition over Inheritance rule. An entity is more than just a list of components, for example: player, tank, bullet; components are data that can be added to any entity, for example: health, velocity, position, fuel, sprite. A system operates on related groups of components, components that belong to the same entity.

The ECS architecture allows greater flexibility in defining entities where every object in a game's scene is an entity, the behavior of an entity can be changed at runtime by adding or removing components. Since components are simple

data buckets, they have no dependencies. The system will always check each entity for its components, and, if the entity has them, all the system will perform its logic on that entity. If not, the entity is simply skipped, with no need for complex dependency trees.

3.1.2 Graphics

The game now uses Slick2D libraries to render the graphics. In terms of aesthetics and functionality, the team felt that it is quite similar to our game's case. However, since Slick2D allows the flexibility of using game states, the switch was then made to Slick2D.

Game assets are sourced from various royalty free third-party websites. The sprites are mainly from https://kenney.nl/:

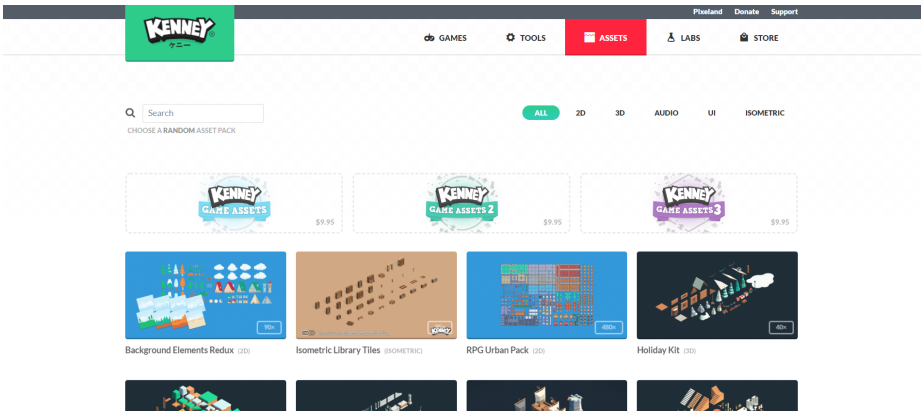

Figure 1: A snapshot of the website where most sprites are obtained from.

Figure 2: Tank sprite used in the game.

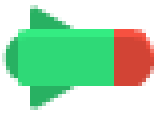

Figure 3: The tank's cannon shell sprite used in the game.

Figure 4: One of the maps in the game when 'Normal' game mode is enabled.

Figure 5: One of the maps in the game when 'Ice' game mode is enabled.

Figure 6: A snapshot of the main menu.

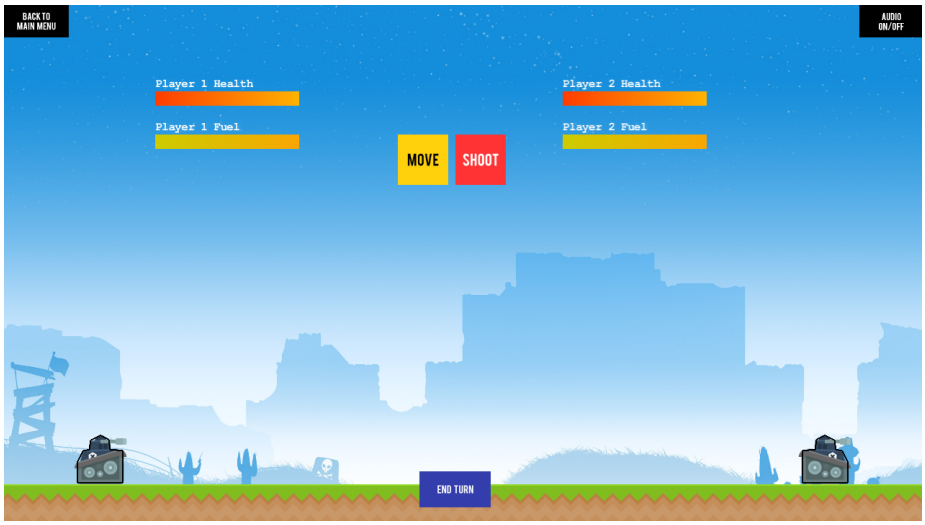

Figure 7: A snapshot of the game in Single Player mode.

3.1.3 Artificial Intelligence (AI)

The Java class that is most concerned with the Artificial Intelligence of the game is AIPlayer.java. This class describes the behaviour for the tank opposing the human player's tank in the single player game mode. It is a child of Abstract-Player and inherits properties from it such as width, height, fuel, velocity, etc... just like HumanPlayer.java does. The AI has 3 levels of difficulty. An increase

in level of difficulty means that the AI-tank would, in principle, be harder to defeat. In practice this means 2 things. Firstly it means that the AI-tank has an improved range meaning that it is able to aim and shoot at its target from a further distance. And secondly, it misses its target less frequently.

The 3 levels of difficulty are:

- EASY(default): The AI-tank has the same range as the human player's tank. It misses its target, on average, 35% of the time.

- MEDIUM: The AI-tank has a slightly longer range than the human player's tank. It misses its target, on average, 20% of the time.

- HARD: The AI-tank has a much larger range than the human player's tank. It misses its target, on average only, 5% of the time.

The AI-tank's behaviour consists of moving or shooting. Seeing as one of the primary limitations of the game on a player is that a player can not shoot and move in a single turn, the AI needs to calculate which of the two it is going to do on its turn. This is determined in the Calculate() method which returns a Boolean value. If it returns false, the AI-tank moves. If it returns true, the AI-tank shoots. The Calculate method is recursive and takes a single argument, the angle. The idea behind it is that it will calculate whether or not is it feasible for the AI-tank to shoot the human player's tank at this particular angle. If it can, return true. If not, change the angle and call the method recursively as such: Calculate(new angle). Essentially, it tries out many angles until it finds one that satisfies the conditions for the method to return true. Instead of slowly incrementing the angle until it finds a correct one, the algorithm takes a divide and conquer approach. It starts with a low angle and increments by a large number of degrees. If the angle is greater than 90 degrees after being incremented, start again with a low angle and increment by a smaller amount of degrees. If no correct angle is found, then eventually the increment will become insignificant enough to neglect and the algorithm returns false as it concludes that the AI-tank can not successfully shoot this turn. This is not the only condition by which Calculate returns false. If the tanks are too close to each other the method returns false. In this case, the AI-tank will choose to move away from the player as opposed to towards it.

Another important function is the isPlayerNearTrajectory() method. When it is said that an angle is "correct" it is meant that the human player's tank is in the way of the current trajectory of the AI-tank's bullet that is calculated based upon the angle passed as an argument in Calculate(), meaning that the bullet would hit its target. This is calculated in isPlayerNearTrajectory().

```
public boolean isPlayerNearTrajectory(double width, double height,
    double centre_x, double centre_y, double i_b_v_x, double i_b_v_y) {

      /*
     the equation below describes the bullet's trajectory.
     */

     y = init_y + i_b_v_y*((centre_x - init_b_x)/i_b_v_x) -
         0.5*g*Math.pow((centre_x-init_b_x)/i_b_v_x, 2);

     /*
     below in the if statement is calculated whether the position of
         the opponent's tank crosses anywhere with the trajectory of
         the bullet
     */

     if(((centre_y +(height/2)) > y) && ((centre_y - (height/2)) > y)) {
        return true;
     }
     else
        return false;
  }
```

3.1.4 Audio

The game includes various audio elements such as background music for the main menu and in game as well as sounds effects for tank movement, bullet collisions, etc. The audio elements were taken from https://opengameart.org/, which is an open-source game assets website that offers game audio and other game elements for non-commercial use.

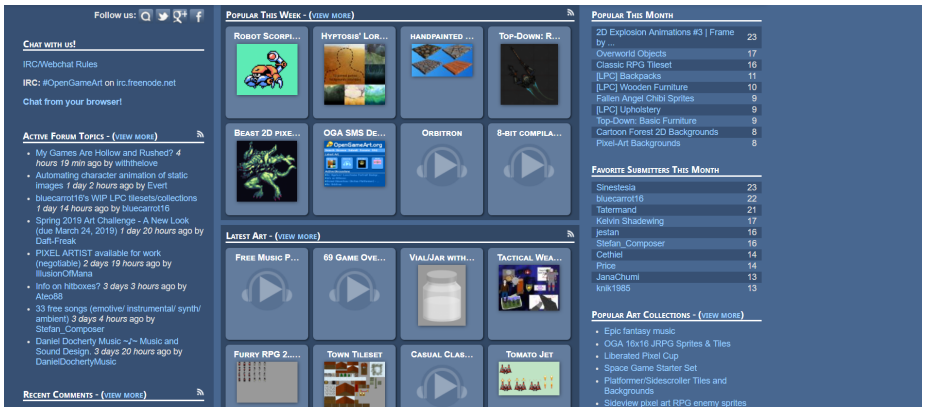

Figure 8: A snapshot of the website where the audio is obtained from.

3.1.5 Physics and Collisions

The following section sets the laws of physics that should apply in this game world.

- Laws of Movement. The movement speed of the player's tank is not affected by the angle of elevation and angle of depression between the tank and the x-axis.
- Only solid objects are affected by gravity.
- There is no friction.
- The tank's movement speed is instantaneous and constant as described above. There is no acceleration. The same rules apply for angular movements.
- Projectiles always have a parabolic trajectory.

The parabolic trajectory of the bullet is calculated based on the following equations.

$$\alpha = 0.7 \tag{1}$$

Horizontal velocity of the bullet:

$$velocity_x = cos(\theta) * \alpha \tag{2}$$

Vertical velocity of the bullet:

$$velocity_y = cos(\theta) * \alpha \tag{3}$$

The initial horizontal and vertical velocity of the bullet is based on θ which is the rotation of the gun barrel's angle of the tank that shot it.

3.1.6 Networking

The game will use the client-server architecture with the server having the authority over the current state of the game. The multiplayer is played in real-time, as opposed to the turn-based single player mode.

The server will run 2 concurrent threads in order to receive commands from the players and broadcast the changes made to the game world to both clients.

Transmission Control Protocol (TCP) will be used for the communication between the server and the client. This is because the game is has only two clients, therefore transmission time is not a major issue.

The client and server communicate using the `BufferedReader` and `PrintWriter`

streams. Previous versions used `DataInputStream` and `DataOutputStream`, as well as `ObjectInputStream` and `ObjectOutputStream` before that.

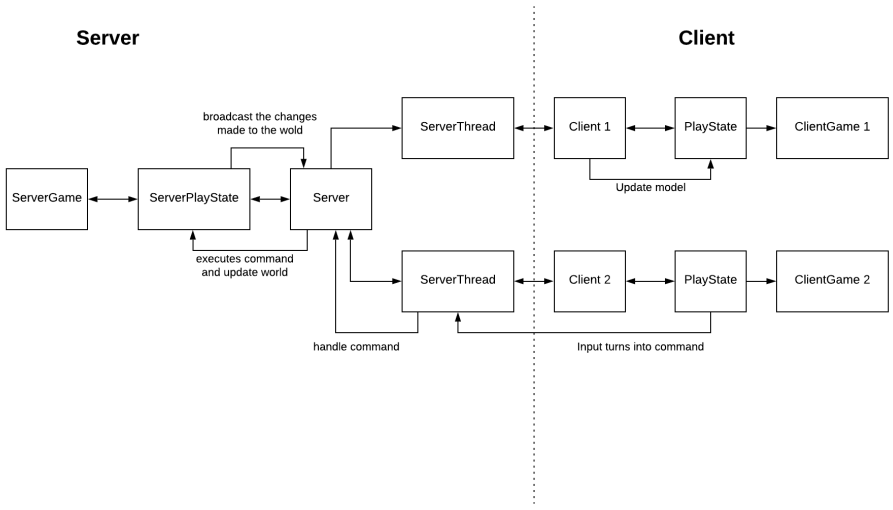

Figure 9: The networking protocol.

3.2 Game Logic

3.2.1 Game Modes

Single Player: The single player game mode allows the user to play against a computer-controlled opponent. In this game mode, input from only one player is expected throughout the game, with the other player being controlled by the Artificial Intelligence (AI). The user can access the single player game mode by navigating to the main menu and left-clicking the “Single Player" button.

Multiplayer: The multiplayer game mode allows the user to play against another human-controlled player. In this game mode input from both players is expected throughout the game. The user can access the multiplayer game mode by navigating to the main menu and left-clicking the “Multiplayer" button. The two players must use two different clients.

3.2.2 Gameplay Mechanics

The game starts with two tanks on each side of the screen. The objective of the game is to make the other player’s health go to zero by shooting bullets that are fired from the tank’s barrel.

For single player, the game is turn-based and each player has to choose to either shoot or move each turn. If the player chooses to move, the player can

move to the right or to the left. While moving, the tank consumes fuel and when the amount of fuel reaches zero, the player's turn will immediately end. If the player chooses to shoot, the player can adjust the barrel of the tank and shoot the bullet. The turn will immediately end if the bullet hits an object, whether it is a player or the ground. For multiplayer, the game is played in real-time and each player can simultaneously shoot and move.

3.2.3 Player Controls

When the match begins, the player can use the mouse to click on the “move" or “shoot" buttons on the screen.

If they choose to move their character, when they press the “left-arrow" or "A" key, the player character's will move to the left.

If they choose to move their character when they press the “right-arrow" or "D" key, the player's character will move to the right.

If the player chooses to shoot, they can then use the “up-arrow" or "W" and “down-arrow" or "S" keys to adjust the gun barrel's angle leftwards and rightwards respectively and the "SPACE" key to fire the projectile.

When the player clicks the “End Turn" button, that player's turn ends and the other player's turn begins.

The “End Turn" button can be clicked at any time during the player's turn.

3.3 Game States

3.3.1 Menu State

The menu state consists of the main menu, which links all the states together. On the main menu there are several buttons the user can click to enter the other states present in the game. The buttons present in the main menu are the Singleplayer, Multiplayer, Tutorial and Exit buttons. The Singleplayer button will enter the Singleplayer game mode, the Multiplayer button will enter the wait state , the Tutorial button will enter the tutorial state and the Exit button will exit the game.

3.3.2 Wait State

The wait state acts as a lobby for the Multiplayer game mode. When the user enters the wait state, the server will be waiting for another client/user to enter the wait state, when 2 players are connected then the Multiplayer game mode will launch. The wait state has a back button on the top left corner to return to the menu state.

3.3.3 Tutorial State

The tutorial state contains information on how to play the game, this includes the mouse/keyboard controls and the rules of the game.The tutorial state has a back button on the top left corner to return to the menu state.

3.3.4 Game Modifiers State

The game modifiers state lets the user choose which game modifiers to turn on/off. The game modifiers state has a back button on the top left corner to return to the menu state.

3.3.5 Play State

The play state contains the actual game itself, and within the play state there are two types of game modes, Singleplayer and Multiplayer. The play state has a back button on the top left corner to return to the menu state.

3.4 Unified Modeling Language (UML)

3.4.1 Simple Class Diagram

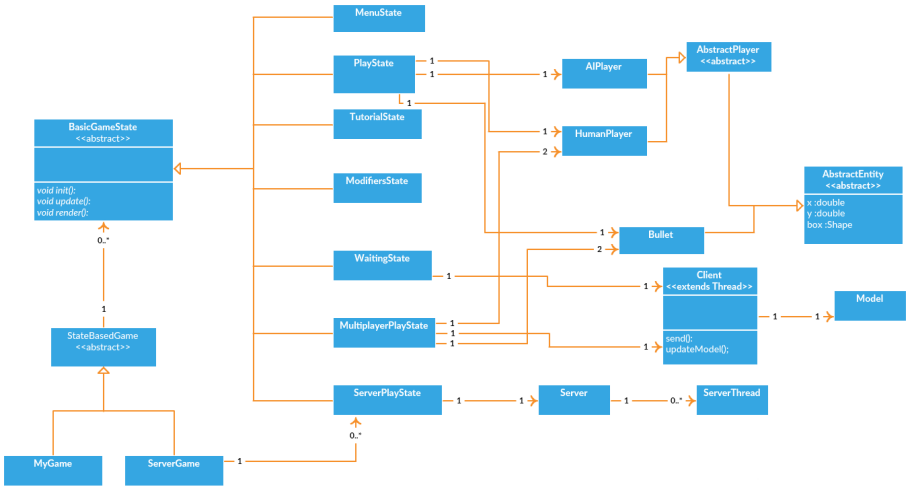

Figure 10: Class Diagram.

4 Interface Design

Human Computer Interaction (HCI), also known as User Experience, represents the research that is taken in designing various User Interfaces (UIs). The team has opted for a simple and clean UI that adheres to the conventional HCI principles, specifically the Norman design principles. The Norman design principles

include visibility, feedback, consistency, mapping, constraints and affordance.

Visibility emphasizes the visible functions that are present in the game. This could be in the form of buttons, prompts and graphics that make the user know what to do next. For example on the main menu if the user wants to enter Singleplayer game mode, they would click the Singleplayer button on the screen to enter the game.

Feedback focuses on the ability of the game to send back information to the user when an action has been taken. This could be in the form of visual, audio or tactile cues that inform the user that the action has been taken. For example if the user chooses to move during a turn and they press either the A or D keys, the tank will move to the left or right respectively.

Mapping refers to the relationship between the controls and effects in the world. Everything needs some sort of mapping, whether it's flashlights on a car or a cockpit in an airplane. In the game's context, an example of this would be when the user presses the right or left arrow keys the tank would move left or right respectively.

Consistency emphasizes on designing applications that have consistent or similar themes, elements and operations throughout the application. An example of this would be a consistent layout that would be used on all the game screens, the same type of font used throughout the whole game etc. This way the user feels familiar with how the UI looks and doesn't have to get used to different layouts, colour schemes, fonts etc.

Affordance is the attribute of an object that allows the user to know how to use it. This is especially important if the application has a lot of on screen elements such as buttons, sliders and check-boxes to name a few. The elements on the screen must be designed in a way that would make the user easily figure out how to use it. For example, buttons in the UI of the game invites the user to use their mouse to click on them.

Adhering to each of these principles ensure that the UI is designed with simplicity and user-friendliness in mind. The menu includes two game mode selection buttons (singleplayer and multiplayer), a tutorial button that the user can press to see how the game is meant to be played and an exit game button that quits the game. There is an audio button in the top-right corner(both in the main menu and in-game) that toggles all of the sound effects on or off. There are no animations in the menu to minimize load times. The team has tried not to over-saturate the game UI to simplify repetitive tasks, so the UI only has the necessary elements that the player needs, this includes the health bar, fuel bar, move/shoot buttons, audio button and the back to menu button.

Slick2D is used to create the UI because the team found that by linking the main

menu states, the game was easier and more intuitive than using JavaFX, which the team has used previously to create the UI. Another reason why Slick2D was chosen over JavaFX was because the team wanted to avoid clashing of two different game libraries, which would possibly lead to incompatibility and failure to link the main menu and the game.

5 Software Engineering

5.0.1 Process Model

Although the team believes that the functional and non-functional requirements of the game is well-defined, the team felt there is a need to create room for flexibility in the event a need arises for requirements to change, and this has proven useful especially in our case which will be further elaborated in the 'Risk Analysis' section. Therefore, the team has decided to adopt the incremental delivery model.

The incremental development model allows flexibility and anticipation of change. The game's requirements are defined early on, however throughout the development stages, the team has evaluated the gameplay mechanics from aspects such as whether a particular feature is intriguing enough for the player.

The team also considered feasibility aspects in terms of the time taken to complete the development of a particular feature. Several changes were made, and the incremental development model allows concurrent activities such as requirements specification and development to take place simultaneously.

Therefore, the team is able to go back and forth between refining the requirements and actually developing the game. That aside, the flexibility allows us to also produce working prototypes in a relatively shorter period of time so the team will be able to review them and obtain feedback from other users.

5.0.2 Software Engineering Principles

The work is based on three fundamental software engineering principles: incremental development, anticipation of change and "You aren't gonna need it" (YAGNI). The game is built in small increments to simplify the validation process. It also eased the handling of future changes in the requirements. If there were any errors detected, then they were partially isolated which made them easier to correct. An example of this would be the networking part of the game. The team met drawbacks when it came to the networking component.

The team then decided to re-create a basic version of the game with just the networking component and the other components such as graphics, AI and audio were removed. The team then made a breakthrough and the game evolved from being a turn-based game to a real-time game. From this example, this

coincides with the YAGNI principle that practices "doing the simplest thing that could possibly work" (DTSTTCPW).

6 Risk Analysis

At the start of the project, during the very first few meetings, the team has conducted Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis to evaluate our strengths and weaknesses and to devise strategies to make the development process more efficient and effective.

Figure 11: SWOT Analysis Diagram.

The diagram was very useful when it came to helping the team to plan our schedule ahead, especially the 'threats' section of the diagram. For example, the complicated AI calculations were handled by two people in the team instead of one. Other similar situations where more challenging aspects of the game development that demanded more manpower, the same strategy of manpower allocation is used. Therefore, the team is able to be on track with the overall progress of the development of the game.

Moreover, the reason why the team chose incremental development over other software engineering processes is because it allowed us to make changes during

development. This is especially relevant in our case because it helped us make major overhauls to our game, such as changing the game's rendering engine from JavaFX to Slick2D.

Another good example of how the incremental development process model has helped us is in the networking section of our game. The team had drawbacks with the networking in the intermediate stages of development. With advice from the project coordinator, the team tried to come up with a minimum viable product that only has the networking component to ensure the logic works, which goes in line with the YAGNI principle.

The process model and the software engineering principles used allowed us to make changes to a major part of our game without halting progress. The team initially started using `ObjectInputStreams()` and `ObjectOutputStreams()` to send and receive to and from the clients.

```
streamIn = new ObjectInputStream(socket.getInputStream());
streamOut = new ObjectOutputStream(socket.getOutputStream());
```

This resulted in a lot of errors that caused development to stagnate for up to a couple of weeks. The team then changed to using `DataInputStreams()` and `DataOutputStreams()` to try and solve the bugs, since they are able to send and receive multiple types of data.

```
streamIn = new DataInputStream(socket.getInputStream());
streamOut = new DataOutputStream(socket.getOutputStream());
```

However, this caused a runtime error due to occasional errors in decrypting the messages by using `DataInputStream()` and `DataOutputStream()`. In the end the team managed to solve the problems by using `BufferedReader` and `PrintWriter`, because the team changed the networking protocol to only send and receive strings.

```
streamIn = new BufferedReader(new
    InputStreamReader(socket.getInputStream()));
streamOut = new PrintWriter(socket.getOutputStream());
```

7 Evaluation

In terms of the team, each member of the team had a specialised area of interest that made it easy to delegate tasks. This also meant that the workload was balanced and no one felt that they were doing too little or too much work. The team was also very knowledgeable in the GUI and handling inputs so when it came to designing and implementing them it wasn't too much of a problem. The team always kept each other updated and had on average two weekly meetings.

The team would meet more often if it was necessary.

However, there were a lot setbacks and the problems that caused massive delays during the development process. One of them was that not all members of the team had prior knowledge of game development, so some of the team were inexperienced when it came to testing and debugging certain aspects of the game such as networking and rendering. The team had to go through a lot of trial and error to get the game up and running smoothly. By far the biggest setback the team had was with the networking. The team had initially planned that both the singleplayer and multiplayer game modes to be turn-based since the team wanted to make both game modes consistent with each other. The team managed to get the singleplayer mode working, however it was with the networking aspect of the multiplayer mode that really halted development for the team. The team had tried for weeks to solve the networking issues for the turn-based multiplayer mode, but the 10th week of the semester arrived and the team still didn't have a working multiplayer mode. At this point, the team was lost and didn't know what to do next. The team overcame this obstacle by taking one day just to discuss and exchange ideas between each other and see which one was the most ideal route to take for the networking. The team eventually agreed to start the networking from scratch and make the multiplayer mode real-time since it was more straightforward to implement. With only over a week left until the presentation, the team scheduled daily meetings to make sure everyone was on the same page and that everyone was contributing to the project. In the end the team managed to get the networking and the multiplayer mode ready for the presentation. This was what the team was most proud of, the fact that the team was able to be resilient in the face of adversity and being able to produce results.

Overall, the team learned how to program a game in Java using the Slick2D game library as well as overcoming obstacles together as a team to achieve our goal. What could have gone better is that the team could have planned the development schedule better and made sure everyone is contributing and doing the work they were supposed to be doing. This would have led to a more polished game with better features and enhancements.

8 Summary

In summary, Warstrike is a 2D side-view artillery turn-based tactics video-game that has a turn-based Singleplayer mode and a real-time Multiplayer mode. Most of the game's elements, such as the GUI, graphics and the audio are rendered using the Slick2D game library. The AI is created using Java's core libraries and the networking uses the TCP/IP networking protocol.

9 Teamwork

The evaluation document is submitted individually on the Canvas page.

10 Individual Reflections

10.1 Anca Burduv

When I first found out that that our assignment was going to be developing a video game I was very excited. I came up with a lot of ideas on how the game could be and so did my team mates. What we ended up with was a collection of exciting and ambitious ideas, some of them more realistic than others. I think this was a primarily good thing because it gave us some flexibility in how we were going to approach the game. However, it also caused a fair deal of indecisiveness, since we took a long time to decide on a final common vision on how our game was going to look like. But I feel like this was not necessarily that big of a drawback, since the team's frequent and effective communication, as well as the excellent team work, lead to a final unified vision for our game and a good understanding of each member's responsibilities. I think we were particularly fortunate to have such a well-rounded team, with each member having a different area of interest.

My personal responsibility on the project was mostly the networking. While I previously had a little exposure to socket programming and concurrency issues, this project has really developed my understanding of different network protocols and how to implement them. Switching from a turn based game to a real time one was something I was very nervous about at first, but now I strongly feel that it was something that should have been done earlier. Because of this, the development halted for almost two weeks trying to fix an error related to the turn based networking. That issue was later sidestepped in the real time networking, resulting in a better architecture and a more appealing game.

Overall, I feel like our team had a good work ethic and interesting ideas, while being held back by poor planing and time management.

10.2 Andrei Mihai

When I found out that the team project module consists of developing a videogame I was very hyped and couldn't wait to meet my teammates and get started with it. I had previous experience in developing games in Unity and game testing so I came up with many ideas during the requirements part of our project which was an advantage for the team, as we had a big selection to choose from, but it also resulted in more time spent on the final idea of the game and how we wanted it to look like. It didn't take us too long to decide which role everyone in the team will have as all of us had different areas of interest, mine being physics,collisions alongside Stefan and game balance. The team showed great

communication abilities and had an open mind when it came to ideas on how we can improve the game. In the first weeks we researched what libraries, software engineering principles and process model to use. After that we started coding and everyone of us had different parts of code, but nothing that can actually be tested, except the GUI. It was Stefan's leadership skills and positive attitude that got us out of the deadlock we were into.

My main role in the team was getting the player movement done and implementing platform obstacles for the single player and multiplayer. I was anxious about my tasks as I had mental health problems during the course of this year and couldn't really concentrate. In the first two weeks everything went very well, I documented on what type of game to make and made the player controls, but afterwards everything got to a halt from my part as I felt overwhelmed. Before the demo I had a discussion with my teammates which were really understanding of my situation and they helped me get back on track and I started fixing different errors that came up during the development phase. After the demo everyone shifted their focus on fixing the networking and we lost at least two weeks on this impasse.

In the end we decided to switch from turn-based multiplayer to real-time, but we had only one week to reconstruct almost everything. It was due to everyone's excellent teamwork and communication that we managed to finish our project on time. We met everyday during this week and everyone had different tasks. Mine was to implement different maps on single player by changing the speed and the fuel consumption of the tanks and adding different platforms depending on the type of map the player chose to play. The latter one's implementation took some time and there was no time left to make the AI's calculations take into consideration the obstacles so we decided to drop this idea.

Overall I am aware that I was not as involved as everyone else during some parts of this project because of my mental health issues and my biggest regret is letting my teammates down at certain point. I can say that I developed my Java programming skills and learned about different libraries and software engineering principles. I am glad that I had teammates that were understanding with my problems and also about the fact that in the final week we worked as hard as possible and managed to finish this project on time.

10.3 Kelvin Chua

It was during the winter break that I first got notified of the Team Project module that was going to be happening this semester. At first, I was pretty anxious since I didn't have prior game development experience at all and was afraid that I wouldn't be able to contribute much to the project. However, all my doubts were cleared when I met my team at the start of this semester.

I already knew one of them since we both came in from Malaysia as part of

the transfer program, so it was great to know that there was at least a person I knew from class. As for the other members it was my first time meeting them and they were really kind and open. We spent our first meeting getting to know each other and instantly there was some chemistry between us. Once we have gotten the icebreaker out of the way, we started focusing on the project at hand, which was to create a game using Java. Luckily two members of the team, Liam and Andrei, has had prior experience of game development so this made me feel less anxious about the project. For the first week, we all researched on what type of game we should make and what Java libraries we should use. In the end, we decided that we would make a 2D turn-based artillery game, similar to that of Worms. As for the game libraries, we would use a combination of Slick2D and JavaFX.

Throughout the project, I was mostly responsible for the GUI, this includes the main menu and other game states. I made sure that the game states are coherently linked with each and other with no errors. I had to adhere to Norman's design principles to make sure that the GUI was visually pleasing as well as easy to use. Since GUI and graphics are closely related, I spent most of my time working with Ken (Weng Chan) to make sure the GUI and the graphics were consistent throughout the game. Whenever I had gaps in between developing the UI I would offer my assistance to the other members, assisting them in completing the other main elements of the game. However, as time went on, it dawned on us that JavaFX would be too difficult to integrate with Slick2D, so during Week 4-5 we decided that the entire game would be developed using the Slick2D game library. We also had a rough time trying to make the networking work for our turn-based game, so in the end we opted to make our Singleplayer mode turn-based and our Multiplayer mode real-time. What I was really most proud of was my team's ability to overcome major obstacles to achieve our goal. This came in the form of my team's struggle with the networking. For weeks we didn't manage to fix the networking for the multiplayer mode and it was already at the end of week 10. My team basically hit a brick wall and didn't know what to do next. So what we did was we took a day just to discuss what to do next and in the end we all agreed to start the networking from scratch and do a real-time multiplayer mode. From this point on my team decided to meet daily to make sure everyone was on the same page and the networking is making steady progress. As a result, we managed to make the networking work just in time for the presentation, and I was pretty proud that we managed to be resilient in the face of adversity.

Overall, this team project has been a challenging yet worthwhile experience. Not only did I get the chance to learn how to create a game in Java, I also got the chance to get a taste of what it is like to work in a team in the real world. I will definitely never forget this opportunity and the friends I have made.

10.4 Liam Skop

As someone who does have previous experience with game development and has been an active member of GDS for the past two years I was excited to jump into the task of making a game although I was wary that I would not be developing in a comfortable environment such as Unity. I might have gotten slightly overexcited initially as I immediately started researching the complications of developing an isometric game. I quickly realised, however, that I would have to meet the expectations of my team and we decided on (the probably wise decision of) making a 2D-game. I wasn't overexcited about the subject of the game as combat games are definitely not my favourite genre. But I decided it would be a good opportunity for me to deviate from my usual genre of RPG/strategy games. From the start, all of us in the team got on fairly well. The first 3 weeks, however, were very disorientating as the lack of a team leader left the team in slight confusion and disarray with regards to who was doing what exactly. There was always someone who was more or less aware of everything that was going on in the project. At an early stage, I was very much on top of how the entire game was put together and how everything was meant to be integrated. When we made the switch to Slick2d, Stefan kind of took the lead on that and became the person who most defined the structure of the program. At that point, as the game was becoming larger and there were more Java classes to keep track of. I shifted my focus to the single-player mode. My primary task was to develop the AI for the game. That duty eventually turned into a more general task of managing the single player game mode. The task was challenging for me because I am not very experienced with physics formulae and writing the AI involved working out several equations required for predicting bullet trajectories. This has been a very positive learning experience, however, and I am glad I got to develop my skills in writing game AI as this will definitely prove useful in the future. I can definitely say I have improved very much as a game developer and so regardless of the issues with the product I am happy with the result. Considering the many issues we have faced as a team, whether it has been certain team members or if it has been particular aspects of the program giving us a lot of trouble (particularly the fact that we've had to completely bin our initial multiplayer mode, under two weeks before the deadline). I am very satisfied with the result and proud of how close we got to meet our original expectations. These kinds of projects rarely turn out the way you initially imagined them but in this case, I am quite amazed that we have managed to stick so closely to our initial vision for the game and I am very impressed with the team and how we managed to really do our best right up until the end.

10.5 Stefan Nedelcu

Being very confident in my programming skills in general and especially in Java, I had few insecurities regarding this assignment even before I met my teammates, most of whom I had already known. I think my optimistic attitude may have inspired others at times, but with a cost, since we eventually hit a brick wall

partially due to a combination of overconfidence, insufficient testing and lack of time.

We got off to a great start as everyone attended our regular meetings and we managed to split to workload evenly based on each person's preferences and skills. This kind of team cohesion helped us focus on our individual tasks and communicate efficiently while also being aware of each other's code development. Thinking retrospectively, however, I admit that we have spent too much time and effort on our original SRS document. Even though it resulted in a high mark which boosted morale, we should have worked more on our system design as we eventually encountered difficulties at every step due to poor planning. Personally, I believe that we had too many incompatible ideas that we were unable to merge into a viable project. As a result, the final version of our game might seem lacklustre and overly simplistic.

Originally, I chose to work on game physics and collisions based on my understanding of how objects should move and interact with each other and my strong mathematical skills. Nonetheless, that did not stop me from contributing to other aspects of our project. If I were to name my role in this team, I feel that it would be problem fixer. Whenever a major issued arose, I came up with a vague idea that eventually morphed into a viable solution through discussion with my teammates and extensive testing.

Overall, developing a video game in Java was an intense experience that enabled me to hone my abilities and extend my knowledge. I managed to finally grasp the intricacies and protocols of building a working communication system using Java networking-related classes, which is something I had struggled with in the past. Moreover, until this assignment came along, I had been reluctant to use Git, although I knew it was good practice and that it offered very useful version control. Now I can honestly say I am more comfortable with Git commands and working on a project alongside more people, skills that will prove themselves to be beneficial to my career.

10.6 Weng Chan

I remember I was still on my last week of the winter break. An announcement was made on Canvas about the Team Project and the brief was released. I took a glance of the document and I was shocked at the amount of work that we had to do. Words such as 'Game Engine' and 'Renderer' were all very new to me. It did not take long for the assignment to become very intimidating. I'm excited to make my very first video game. On the pessimistic side, I am a direct entry student. Although my first year covered the very basics of Java, it did not cover content such as sockets (which would be used in networking) and creating a graphical user interface (GUI). I was afraid that I would be a burden to the team being someone who learnt a relatively lesser amount of syllabus and has no prior game development experience whatsoever. Thankfully, I met a team who are really considerate. They made me realise that I'm not alone when

it comes to making a game with no experience. After all, we are all here to learn.

Although at that point in time, I have only worked with console applications on Java, I took up the challenge of handling the graphics part of the game. One of my first tasks was to render the graphics such as the tank sprites on the game screen. My first obstacle occurred during Week 2 when I was struggling to work with JavaFX's libraries. I was able to draw the basic components such as the tanks but they are all static and essentially non-functional. The development on the graphics part was a standstill throughout Week 3 and Week 4, but thankfully my team reassured me that it's totally normal and encouraged me to continue. Thankfully, when we made the switch over to Slick2D, the development process had a breakthrough and became a lot more straightforward. I was able to create the essential features needed by the game such as movable objects and background layering with Slick2D's simplicity. One of my main goals when I worked on graphics was to ensure that the game looked aesthetically-pleasing which I have managed to achieve.

Since graphics and the GUI goes hand-in-hand, Kelvin (who worked on GUI) and I worked closely together in making sure the graphical and GUI components throughout the game were consistent in order to adhere to the 'Consistency' principle of Don Norman's Principles of Interaction Design. I have also worked on the audio component of the game. Suitable royalty-free soundtrack for the context of the game was difficult to obtain, therefore I needed to get multiple audio files, mix and match them through Audacity to get a combined soundtrack that achieves a feel suitable to the game. Since the aesthetics also play a major role in the final presentation, I've took the lead to design the presentation slides and the storyboard so that it shares the same aesthetic consistency as the game, with Liam assisting me on outlining the points.

I have learnt that trying something we're not familiar with can be scary, but we can always view it as an opportunity to learn something new. I also realised that teamwork is a really crucial aspect of the project. This is vital especially when it comes to dealing with Git, which can potentially cause a lot of merge conflicts when the team does not communicate well on what they were working on. We also had drawbacks on the networking component, but tough times like these have brought the team together and we have made significant progress in a relatively shorter period of time by communicating more and looking out for each other.

11 Compulsory Appendix

The team has adhered to standard and conventional coding standards during the development of the game. The team has ensured to give proper naming to our variables, methods, classes, packages, and interfaces so that it is easily understandable.

The name of the classes are nouns and the first letter is always capitalized. The name of the methods are verbs, and the first letter is always lowercase. The name of the variables are concise and named in such a way that it describes its own potential functionality and behaviour during runtime. The name of the class and ANSI constants are all in uppercase. Constants in the game are declared as `static` and `final`.

Four units of space are used for every indentation.

For the getter methods, the team adhered to the following conventions:

- set as a `public` method
- the method name should be prefixed with `get`
- the method does not take any arguments

As for the setter methods, the team adhered to the following conventions:

- set as a `public` method
- the method name should be prefixed with `set`
- the method takes an argument depending on the context
- return type should be `void`

Comments are written and the code is annotated with Javadoc comments with the /**...*/ operator. An example of the Javadoc comments made:

```
   /**
* <h2>Play State</h2>
* The play state defines all the in-game logic.
* @author Anca Burduv, Andrei Mihai, Kelvin Chua, Liam Skop, Stefan
    Nedelcu, Weng Chan
* @version 1.2.1
* @since  29-03-2019
*/
public class PlayState extends BasicGameState {
   ...
   }
```

The source files also have the following information, in order:

- Package statement
- Import statements
- Exactly one top-level class

The team also adopted other coding standards that are documented in this guide: https://google.github.io/styleguide/javaguide.html

12 Secondary Appendix

12.1 Secondary Appendix A: Changes from the System Requirements Specification (SRS)

- The main libraries are now from Slick2D instead of JavaFX.
- The system architecture switched from Model View Controller (MVC) system architecture to a Entity Component Systems (ECS) system architecture.
- The turn-based multiplayer game changed to a real-time game.
- Three difficulty levels are added for the Artificial Intelligence (AI) player which are easy, medium and hard.
- Platforms are added in multiplayer mode that block the bullets so that players now will need to play and use their resources more strategically.

13 Glossary

For all intents and purposes, the following terms and their corresponding meanings will be used throughout this document to describe the game.

- Aim: adjust the angle at which the barrel is oriented. The angular speed is 20 degrees per second.
- Collision: a visual representation of the hypothetical physical interaction between two entities; in other words, two entities collide when they touch each other, but their visual representations cannot overlap.
- Ethereal: not solid.
- Gravity: a simulation of Earth's gravity at a vertical acceleration of -10 pixels/s^2.
- Health: a fixed number of hit points that the player can lose before the player's sprite is destroyed.
- Move (regarding player movement): the sprite shifts to the left or to the right following the terrain border.
- Power: an index that is directly proportional to the initial velocity of the projectile. Power has no effect on how much damage is inflicted.
- Projectile: an object that can be shot which travels along a parabolic trajectory until it collides with a solid object. Projectiles are affected by gravity and wind.

- Shoot: when a player shoots, a projectile comes out of the superior end of the tank's gun barrel. Projectiles are solid and affected by gravity.
- Solid: an entity/object/component is solid if it supports collisions with other solid entities. By definition, the terrain is solid. Other solid components include player characters, projectiles and mines.
- Velocity: the number of pixels displaced over time.

References

[1] Google Java Style Guide,
`https://google.github.io/styleguide/javaguide.html`