Bilkent University

Department of Computer Engineering

Object-Oriented Software Engineering

EmojiStrike

Analysis Report

Project Group Ali Altaf Salemwala, Bora Bardük, Eren Çalık, Kıvanç Gümüş

Supervisor: Bora Güngören

Analysis/Draft Report
February 25, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319/1.

Contents

Analysis Report

EmojiStrike

1 Introduction

1.1 Purpose

This document will present a detailed description of the EmojiStrike game. It will introduce the purpose and scope of this game, followed by a detailing of the features, requirements, and constraints of the product. The document will also contain cases which explain how the game system will react to stimuli from the players. The intention of this Software Requirements Specification document is to serve as a future reference point for the developers and the clients, as well as a map for the future development of the project.

1.2 Scope of Project

This game is a multiplayer turn-based game, shown to the user as a 2-D map on the monitor screen. The game consists of avatars belonging to the players dropped in random places on a pre-defined map, followed by players then attacking each other with ranged and un-ranged weapons, with the last remaining player being the victor. Power-ups designed to enhance gameplay (for example, by making a player temporarily invincible, or increasing the potency of their weapons, or increasing a wounded player's health) will be added to the map at certain time periods during the game.

The system will be designed to maintain player interest and interactivity in the game, as it is intended as an object of leisure. It will attempt to maintain player participation until the end of the game and will provide challenges to the player while also maintaining a record of previous victors, to foster competition.

1.3 Glossary

Table 1 Glossary used for EmojiStrike

Term	Definition
Power Up	Objects which boost the abilities of the player that captures them.
Avatar	The image used by the player to represent them.

1.4 References

IEEE. IEEE Std 830-1998 IEEE Recommended Practice for Software Requirements Specifications. IEEE Computer Society, 1998.

1.5 Overview of Document

The Overall Description section of this document gives an overview of the functionality of the game. It describes the informal requirements and is used to establish a context for the technical requirements specification section that follows.

The Requirements Specification section of this document is intended primarily for the developers, and so describes in technical terms the details of the functionality of the product. Both sections of the document describe the same game, but since they are intended for different audiences, different language has been used.

2 Overall Description

2.1 Product Perspective

Our product is self-contained, and standalone in terms of functionality. There are no external data servers required since it will be a turn-based game on the same machine. Our main actor is the players who are going to play the game.

2.1.1 System Interfaces

The user will use the keyboard to pass in inputs. While selecting an Emoji avatar for each player, and the map of the game, players will use the four arrow keys to navigate the menu, the Enter key to select, and the Backspace key to move back. The user will use the letter and number keys to type in their character's name. During gameplay, the W, A, S, D keys will be used to move the character, and the four arrow keys will be used to aim the weapon, while the Space Bar will be used to fire.

The "P" button will be used to move the game in and out of the Pause mode. The "R" button will be used to restart the game and bring it back to the main menu, and the "Q" button will be used to quit the game. "Shift"+ "Q" saves the game first and then quits. A dialog box will open asking the user to type in a name for the saved game.

The outputs will be passed using the computer monitor and speakers. The user interface display will be visible on the monitor, and game sounds will be audible through the speakers.

The game will be saved in a sorted order according to the date. The file which will be used will be in the root directory of the game and will be a CSV file to keep track of different data easily. The data will consist of the game state, date, and desired name to each save file.

2.1.2 User Interface

The game will be displayed in the maximum windowed resolution. In the top-left corner will be a menu with options to pause, quit, quit + save, and restart the game. Next to the commands will be the letters which should be pressed to bring the user into the desired state. The rest of the available space will be occupied by the game.

During each move, a small circular timer will display the amount of time remaining for the player currently doing their move.

When beginning the game, the users will select the number of players. Following this, they will select the Emoji and set the name for each player in sequence, and then finally select the map they will play on, after which they game will start.

The users will select Emojis from a list, and type in the name for each player. They will also select the number of players from a list of pre-approved numbers. A table of maps with small previews will allow them to select the map.

If a player chooses to load the game, there will appear a list of files which the user may load the game from. While saving, there will be a text input box where the user types in the saved game's name and press Enter to quit.

Below are some tentative mockups created for the application user interface.:

Figure 1 Login Screen

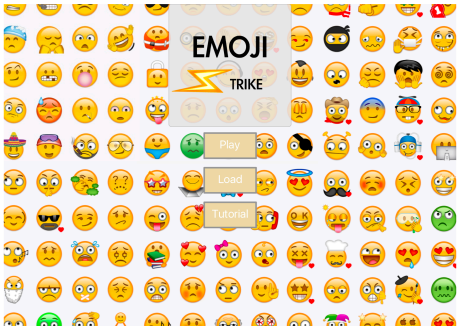


Figure 2 Map Select Screen

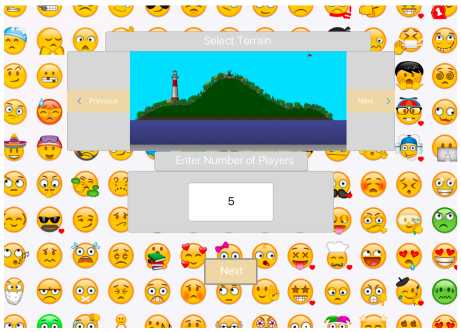


Figure 3 Player Select Screen

Figure 4 Game Screen

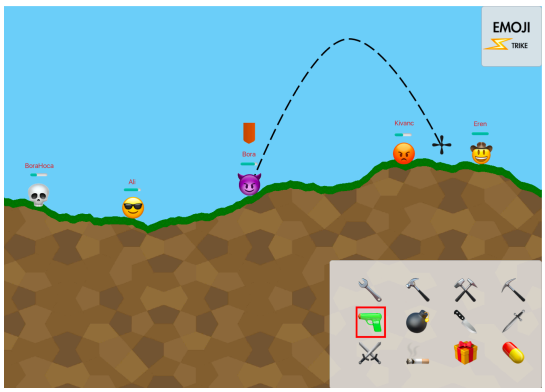


Figure 5 Pause Screen

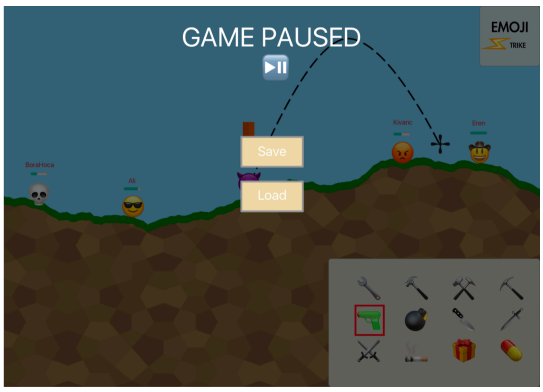


Figure 6 Save/Load Screen

2.1.3 Hardware Interfaces

Our program will use Java's automated HID USB device recognition to connect our application to hardware. The required keyboard, sound, and display interface will be detected automatically without the manual utilization of physical ports. Java will interpret the signals automatically and convert them to Java Virtual Machine's inputs. The supported devices matter on the port availability between a computer and its physical port interfaces. It can be USB-C, USB-A, HDMI, etc.

2.1.4 Software Interface

The game will be written in Java language and will interact with an operating system of any platform using the Java Virtual Machine. The Java specifications are:

- Java Version 8 Update 121 from www.java.com
- Communications Interface
- N/A

2.1.5 Memory Constraints

In the Java Virtual Machine, -Xmx is 2 gigabytes, and the -Xms is 256 megabytes. The program will use the default settings.

2.1.6 Operations

There are four modes of operation. There is the "Loading" mode, the "Main Menu" mode, the "Pause" mode, the "Game" mode.

During the "Loading" mode, no keys can be used. There are no user operations in this mode since the program is loading the game.

Throughout all the following modes, the users can use the "Q" key to quit the game, and the "R" key to bring them back to the main menu and delete any progress.

In the "Main Menu" mode, the user can use the arrow keys to navigate the menu, the Enter key to select, and the Backspace to move them back in the menu. Apart from these keys and "R" and "Q", all other keys are deactivated.

In the "Pause" mode, the user can only use the "R", "P", "Q", and "Shift" + "Q" keys. The "P" key moves the user back into the "Game mode".

In the "Game" mode, the user can use all available keys. Arrow keys will change the weapon's aim, while the "W", "A", "S", "D" keys will navigate the player's character. The "P" key will move the game into "Pause" mode, and the "R", "Q", and "Shift + Q" keys will keep their functionality. The Space Bar will shoot the weapon.

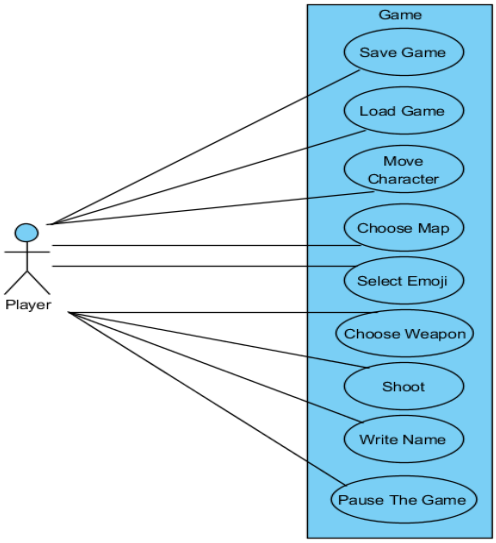
2.1.7 Site Adaptation Requirements

a) The data initialization will be done in the same way for every deployment since there will be no difference when the application will be deployed into identical Java Virtual Machines. During the menu state, the texture files and menu music will be retrieved from the aspects file of the game. During the play state, the map and player figures will be rendered from the selected pre-existing map file before the game starts. There will be additional components which will procedurally be required, like sound and visual effect files. This data will be rendered real-time since their timing and locations are unknown beforehand.

b) Since our application will not change functionality in different areas of deployment, we will not have any specific mission-related features.

2.2 Product Functions

Figure 7 Use Case Diagram



The program will allow users to start the game, pick an avatar for themselves in that game, and choose a map to play on. While playing, users may choose a weapon to attack other players with, attack other players with the weapon, and move around on the map. Players may save the current game if it is incomplete, and load earlier game instances to play.

Save Game: Player saves the game state to be retrieved later. User can rename their save files. By default, date of saving time will be added.

Load Game: Player loads the previously saved game state.

Move Character: Makes the character initiate physical interactions with its surroundings. Player may change character position on the map during this case.

Shoot: Allows the player to attack inactive players. During this case, player can not initiate movement.

Choose Map: Player chooses one of predetermined maps to be used in game case.

Select Emoji: Players choose their avatars, which will be used by individual players during game case.

Write Name: Players choose their names, which will be used by individual players during game case.

Pause the Game: Player pauses the game to be resumed by the player at a desired time.

2.3 Use Case Scenarios

Once all users have registered their characters and the game has started, the system can be in the game state.

Turns will cycle through the list of players. The player with the current turn will have 30 seconds until the turn switches onto the next player in the list.

2.3.1 Move Scenario

To show this scenario, we will assume the player pressed the "A" key to move to the left. If the player is on an open ground in the map, the displayed character shall move one position to the left of its current position.

2.3.2 Cornered Scenario

To show this scenario, we will assume the player pressed the "A" key to move to the left. If the player is on the map but immediately to the left of the player is the end of the map, then pressing the A key will make no difference and the displayed character will remain at its current position. This applies if the player presses the W key and is at the top of the map, S key and at the bottom, and D key if at the right.

The player may also be cornered if they are in a position on the map adjacent to an obstruction (such as a big boulder on a nature map), and try to move in the direction of the obstruction.

2.3.3 Attack Scenario

When a player is the active player, they may attack another player. They will have a list of weapons provided to them, and they may cycle to the next available weapon using the C key. A weapon will be made unavailable if it has no ammo. The player may adjust where they are aiming using the arrow keys, so if they press the right arrow key the weapon will be aimed to the right, and then pressing the up key will incline the aim. The left and right keys will change the aim to their own respective directions. The aim is represented onscreen by an arrow displayed by the system. The user then presses the space key to shoot (he will stab if a knife is selected), and the weapon will fire. The intensity of the fire will depend on how long the space key was pressed, with a longer press resulting in a higher intensity. This action will decrease the weapon's ammo and will also end the player's turn. The fired projectile, if the weapon is a gun, will land according to where the user aimed it and the amount of intensity used, and will damage everything around it by decreasing the health of the struck player and destroyinh the adjacent destroyable map. A player has no option to defend themselves from an attack during another player's turn.

2.3.4 Die Scenario

When a player has been hit by a weapon, damage decreases the player's health. If the damage delivered is greater than the player's health, the player will die. The player's character will disappear from the map, and the character will be removed from the available characters.

2.3.5 Power-Up Scenario

This scenario is triggered if a player enters a position where a power-up icon is placed. The player-up icon disappears and the player's statistics display the change brought about, if there is any such change. For example, an increase in health will increase the health bar of the player, while an ammo increase will show no explicit change.

2.4 User Characteristics

The game requires the user to have basic knowledge on how to use a keyboard. Other than that, no technical expertise, special knowledge, or extensive experience is required. The game is designed to be intuitive and easy to understand for new players.

2.5 Constraints

Developers must create the game in Java.

2.6 Assumptions and Dependencies

We are expecting that the game is designed for computers with regular functioning keyboards with ISO or US layout which possesses the required keys for users to communicate the game.

We are expecting that the game is designed for multi-color and multi-dimensional screens with a size such that gameplay is user-friendly.

We are expecting that the game is developed using an up-to-date Java version to avoid version conflicts.

2.7 Apportioning the Requirements

If the up-to-date Java version of the development time requires different hardware and software constraints (like macOS losing support), requirements may change at the time.

If the up-to-date computer communication technologies render the use of a keyboard, speaker, or display interfaces obsolete; different types of I/O with the user should be implemented.

3 Specific Requirements

3.1 External Interface Requirements

3.1.1 User Interfaces

The system starts a window in Java and waits for keyboard inputs from the user to change the game state. Different commands work in different states, and so not all keys work in every part of the game. For example, the ASDW commands will not work for the menu state.

3.1.2 Hardware Interfaces

The three pieces of hardware in the system will be a keyboard, speakers, and a monitor. The keyboard will allow users to enter commands to be interpreted by the game, and the changes will work in real-time. The speakers will play sounds related to the gameplay in order to create a better user experience. The monitor will display the game to the user. The monitors and speakers will change depending on the input from the keyboard. Both of them will also perform in real-time.

3.2 Functional Requirements

Table 2 Keys and their functions

Arrow Keys	When in the Menu, Load and Choice states, the system shall navigate to the selected

	option from those that are available. The up key will move it up, the down key will move it down. It will loop through the options. In the Load State, if there are no saved games, no action shall be taken. When in the Game state, the system shall adjust the direction where the weapon will be fired, and represent an arrow in the direction (showed by a combination of the keys). When in the Tutorial state, the system shall scroll up or down through the tutorial depending on which key is pressed.
Enter Key	When in the Menu, Choice, and Load states, the system shall execute the selected option. When in the Save state, the system shall make a new file with the data of the current game along with the name entered by the player, and then return the game to its previous state
Space Key	When in the Game state, the system shall fire the selected weapon and trigger the appropriate animations and sounds, calculate any updates, and display them to the screen (such as characters hurt/dying or terrain being destroyed).
Q Key	When in the Game, Load, Pause, and Tutorial states, the system shall terminate. NOTE: we had this for the Save state as well but that's inappropriate... what if someone writes name including Q .... We cannot have letter keys and q keys used in same state.
P Key	When in the Game state, the system shall enter the Pause State. When in the Pause state, the system shall enter the Game State.
R Key	When in the Pause and Game states, the system shall quit the current state and the current game and move to the Menu state.
S Key	When in the game state, the system shall enter the save state.
C Key	When in the game state, the system shall cycle to the next weapon available.

Backspace Key	When in the Tutorial and Load states the system shall move back into the Menu state.
WASD Keys	When in the Game state, the system updates the location of the active player on the map by a predetermined move amount for the character (this amount may change based on power-ups acquired by the character). The A key moves the character left on the map, the D key right, the S key down, and the W key up. If there is no space to move (e.g. if the character is against a wall to the left and presses the A key) nothing happens. The changes shall appear real-time on the screen. In addition to detecting collisions, the system shall also detect if the character moves off a cliff and will calculate and display the appropriate falling motion.
Letter and Number Keys	When in the Save state, the system shall store the input from the user with these keys and save the game in a file with the name. When in the Choice state (in the Emoji selection window), the system shall store the input from the user with these keys and save that user's character with the name.
Shift + Q Keys	When in the Game state, the system shall first open the Save state, then terminate itself directly.

3.3 Performance Requirements

The game consists of only one terminal. The game is played only on one computer, with no outside communication.

The game supports anywhere from one to ten simultaneous players.

Keyboard input will be used to alter the state of the game, which will be reflected via changes in monitor and speakers.

Data transactions will hopefully be handled in real-time since this system is rather light. The user should not have to wait for the game.

3.4 Design Constraints

Limitations imposed by Javax or Swing, for the GUI.

Limitations imposed by the Java Sound API.

Limitations imposed by the Java KeyboardListener used by the GUI component.

3.5 Software Design Attributes

3.5.1 Reliability

The client should have a Java Runtime Environment 8 installed since the system will run on that. The reliability of the system depends on the JVM, and so it should not crash or hang for any reason other than operating system failure.

3.5.2 Availability

The system is available to everyone with Java Runtime Environment.

3.5.3 Security

No need for security.

3.5.4 Maintainability

Software has no real maintainability requirements.

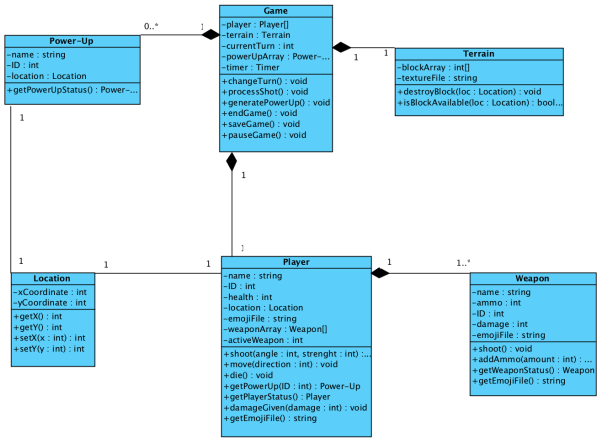
3.5.5 Portability

Since it will run on the Java Virtual Machine, the system is highly portable. It has no other portability considerations.

4 Appendixes

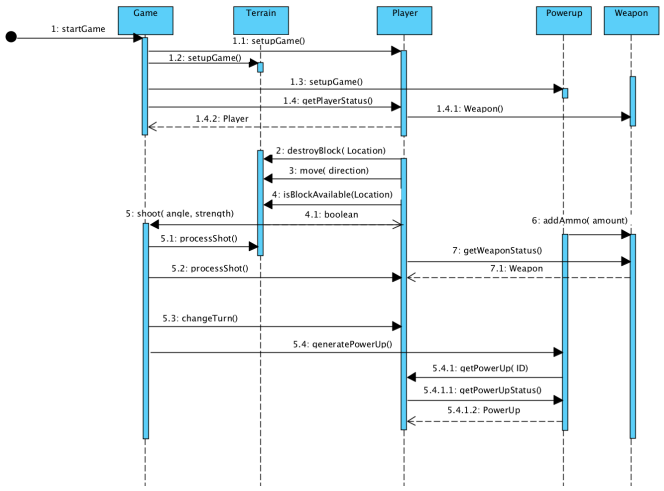
Here is the UML Diagram of our main game functionality objects:

Figure 8 UML Diagram



Here is the Sequence Diagram illustrating interactions between main game functionality sequences:

Figure 9 Sequence Diagram



5 Index

//We will update this part in future revisions