TECHNICAL REPORT

WIX 1002 - Fundamentals of Programming
Session - 2023/2024
Semester - 01

Instructor Name: Dr. Burhan Ul Islam Khan

NAME	MATRIC NO
Anas Abdurahman Mohammad	23055727
Saad Ahmed Pathan	22114077
Christine Leow Si Ting	23004965
Raisha Haque	22111401
Muhammad Ikmal Haikal Bin Mohd Izahan	22001761

CONTENT

Requirements and Explanation of the assigned project	**03**
Approach to solve the task	**04 - 06**
Description for the solution	**07 - 18**
Report Diagrams	**19 - 20**
Sample Outputs	**21 - 30**

Requirements and Explanation of the assigned project

The assigned project for the WIX1002 - Fundamental of Programming (FOP) course is an RPG based text adventure game developed with Command Line Interface (CLI) or Graphical User Interface (GUI). This game requires a combination of technical skills and creativity, demanding both programming insight and imaginative design. The game involves navigating a virtual environment, interacting with different elements, and progressing through various challenges. To successfully complete this project, we were required to must have demonstrate proficiency in Object-Oriented Programming, utilizing Java as the primary language.

Key requirements for the project include:

1. Designing a 40x40 map with navigable spaces and obstacles.
2. Creating five unique archetypes, each with specific attributes loaded from a file.
3. Implementing a levelling-up system to enhance player progression.
4. Developing a monster class, along with specific monster types, to challenge players.
5. Crafting spells for each archetype, adding strategic depth to gameplay.
6. Establishing a round-based battle system for player-monster interactions.
7. Incorporating a save game feature for preserving game progress.
8. Handling abnormal inputs gracefully to ensure a robust user experience.
9. Enhancing the game's visual appeal with colourful text and ASCII art.
10. Utilizing a database for data management and retrieval.
11. Applying Git and GitHub for version control and collaboration.

This assignment not only tests technical skills in Java programming and database management but also encourages creative problem-solving and design. We were expected to go beyond the basic requirements, showcasing our individuality and innovation in their game design.

Approach to solve the task

To solve the "FOP Valley" programming assignment, we have divided this into various parts. The project folder name is Chillcoders as our group name. It includes eleven packages under the source package where each package contains some java classes or important files. Here is a brief description of every class with their respective packages according to the structure of the project

Chillcoders

game.engine (Contains the classes related to the framework of the game)

i. Boundaries.java
ii. CommandLineInputHandler.java
iii. Game.java [Main Class]
iv. Keypress.java

Combat (Contains the classes used for combat mechanics)

i. Combat.java
ii. Spells.java

Entity (Contains the classes for player and monsters, and archetypes)

i. Character.java
ii. Major.java
iii. Monster.java
iv. Player.java

entity.monsters (Contains the child classes of the Monster class)

i. Dragon.java
ii. Gnoll.java
iii. Goblin.java
iv. Harpy.java
v. Ogre.java
vi. Skeleton.java
vii. Witch.java

game.io (Handles the file input/output functionalities)

i. Reader.java
ii. SaveDB.java

game.ui (Responsible for printing out images and texts)

i. Map.java
ii. Print.java

resource.ascii (Does not hold any classes, contains all ascii art used by the game)

i. ai_ascii.txt
ii. book_ascii.txt
iii. credit_ascii.txt
iv. credits_ascii.txt
v. csn_ascii.txt
vi. dragon_ascii.txt
vii. epilogue3n_ascii.txt
viii. epilogue3y_ascii.txt
ix. epilogue4_ascii.txt
x. epilogue_ascii.txt
xi. gnoll_ascii.txt
xii. goblin_ascii.txt
xiii. harpy_ascii.txt
xiv. help_ascii.txt
xv. help1_ascii.txt
xvi. is_ascii.txt
xvii. load_ascii.txt
xviii. lose_ascii.txt
xix. menu_ascii.txt
xx. mmc_ascii.txt
xxi. ogre_ascii.txt
xxii. portal_ascii.txt
xxiii. se_ascii.txt
xxiv. skeleton_ascii.txt
xxv. story1_ascii.txt
xxvi. story2_ascii.txt
xxvii. story3_ascii.txt
xxviii. story4_ascii.txt
xxix. story_ascii.txt

xxx. win_ascii.txt
xxxi. witch_ascii.txt

resource.desc (Contains all the info for map design, archetypes, monsters, and spells)

i. majorsDesc.txt
ii. map.txt
iii. monstersDesc.txt
iv. spellsDesc.txt

resource.database (Contains the database used for save/load game functionality)

i. save.db

resource.images (Contains all the images used in the map)

i. ascii-text-art.png
ii. boy_down_1_1.png
iii. dragon.png
iv. gnoll.png
v. goblin.png
vi. harpy.png
vii. ogre.png
viii. skeleton.png
ix. tile_1.png
x. witch.png

tile (Responsible for printing out the tiles on the map)

i. Tile.java
ii. TileType.java

Description for the solution

To complete the development of the RPG based text adventure game in Java for our assignment, we implemented the following:

Map Design

Designing a 40x40 map consisting of the character, ‘#’ as obstacles where players cannot pass through, and enabling players can move with their characters using the W, A, S, and D keys. Here, W means upward, S means downward, A means left, and D means right.

Archetype Creation

Defining five archetypes by creating classes for each that we termed “majors”. Representing each archetype along with their initial attributes, including health points, attack, defence, and the scaling factor for the three attributes for every level up.

Applying the knowledge of File I/O to handle the archetypes file instead of hardcoding it.

Levelling-Up System

Implementing a system where characters gain "credits" as a substitute for the traditional levelling system. The reason being is "credits" are a more suitable term for the concept of our game. However, it acts the same as regular levels except for needing "experience points" to increase it.

Monster Class Development

Creating a base monster class with the following properties like health points, attack, defence, and credits. After that, deriving specific monster types from it and each type should have unique characteristics. Using the description found in monsters file, creating seven distinct monsters that expand the Monster class. The monsters' locations on the map are generated at random. When a player spawns seven distinct types of monsters will appear.

Spell Crafting

Creating three distinct spells for the various character archetypes, each unlocking at a specific character level and featuring individual cooldown periods. These spells, along with their descriptions, level requirements, and cooldown times, should be documented in a file. Additionally, ensure that the spell designs align with the characteristics of each archetype while maintaining creative freedom for uniqueness and balance.

Battle System

In this text-based RPG battle system, players engage in round-based battles, where they can choose from various moves such as attacking, defending, healing, or escaping. Spells are also available for use. Battles continue until a win, loss, or player exit. The interface provides descriptive feedback. It displays HP and stat details for both the player and the monster. The monster automatically battles, focusing only on attacks without ever defending or healing.

Save Game Feature

To enhance the player experience, a save game feature is almost always essential. Players should be able to record their current game status, facilitating the resumption of progress later. Implementing an automatic saving functionality. Incorporating a database and leveraging File I/O knowledge to create a reliable platform for players to save their game, ensuring a seamless and enjoyable gaming experience.

Input Handling

Ensure the program can handle unexpected or incorrect inputs without crashing. In our game, it is crucial to handle both unintentional and intentional mistakes in player inputs effectively. The system completely ignores any invalid inputs from the user, while the User Interface highlights the valid choices the user can pick from using colourful text.

Colourful Texts and Visual Enhancements

Using ASCII art and coloured text for an engaging user interface. Enhancing the game's visual experience, we can implement color-coding for specific keywords, such as valid user inputs in green and text in red signifying choices that are currently unavailable. This dynamic visual cue improves clarity and player engagement, making valuable information stand out effectively, thereby enhancing gameplay and user interaction.

To elevate the game's visual appeal and combat potential text-based boredom, we can incorporate ASCII art stored in .txt files, rather than hardcoding it into the code. This creative addition adds engaging visual elements to the game, bringing characters, scenes, and moments to life, enhancing the overall gaming experience by making it more attractive and immersive.

We always tried to apply Object-Oriented principles throughout the development process for a robust and maintainable codebase.

game.engine

i. Boundaries.java

This class defines a method called "limitmovement" that is used to control the movement boundaries of an object within a specific area. It guarantees that the object remains within the defined boundaries by adjusting the object's position ('x' and 'y') and speed when it reaches the limits of the specified area.

ii. CommandLineInputHandler.java

This class serves as a KeyListener, capturing and processing user input for a game interface. It manages different progress states, checks input against the game's progress status, navigates through game screens, triggers events, and changes states accordingly.

iii. Game.java [Main Class]

This class serves as the primary driver for a game engine, handling the creation and management of various game screens such as the title screen, character creation, combat sequences, and more. It also manages user input through the "CommandLineInputHandler" class, orchestrates progress states, and navigates between different game stages based on user input and game progression. Additionally, it handles game initialization, loading and saving game states, monster interactions, map navigation, and UI visibility toggles throughout the game.

iv. Keypress.java

This class is responsible for detecting keyboard inputs to control character movement within the game. In this class, it implements the “KeyListener” interface to capture and interpret key presses, specifically the arrow keys (W, A, S, D) for directional movement.

Combat

i. Combat.java

This class is responsible for managing the combat mechanics in the game, including interactions between the player and various monsters, as well as initiating and resolving combats. It contains several methods for calculating damage, defending, healing, attacking, running away, and casting spells. Additionally, it reduces cooldowns for various actions and manages combat-related settings such as defending status, blocking, and cooldowns. It initiates specific combat actions based on player input or game progress.

ii. Spells.java

This class defines the properties and attributes of spells used in combat within the game. It stores information about a particular spell, including its name, type, description, cooldown duration, required credits, current countdown status, and damage multiplier or effect strength multiplier. This class serves as a blueprint for individual spell objects used during the combat scenario. The constructor initializes these attributes by reading information from a file related to the specific spell.

Entity

i. Character.java

This class defines the fundamental attributes and functionalities that are common to all game characters. It includes properties such as position (‘x’ and ‘y’ coordinates), speed, chosen major, health points (hp), attack power, defence strength, available credits, and the character’s name. This class serves as the foundational structure for various characters in the game, providing a framework for their movement, statistics, and basic information.

ii. Major.java

In this class, it encapsulates the information about a specific archetype/major within the game. It holds attributes such as the major’s name, ASCII art representation, description, identifier, and several statistics like health points(hp), attack, defence, and followed by their respective scaling factors. This class acts as a container for major-related data, enabling the instantiation and retrieval of major-specific attributes and spells from the text file that holds all the information.

iii. Monster.java

This class serves as a blueprint in creating and managing in-game monsters. It stores essential attributes such as whether the monster is alive or dead(‘isDead’), the monster’s ASCII art representation, name, description, unique ID, health points(hp), attack power, defence strength, credits, and attack dialogues. This class enables the retrieval of various monster-related data like description, ASCII art, and name. It also includes methods to fetch information from external files and handle the despawning of monsters within the game map.

iv. Player.java

This class defines the behaviour and attributes of the game's player character. It includes functions for character movement, collision detection, drawing the character on the game screen, and handling player-related data such as health points (hp), attack power, defence, credits, and interactions with the game environment. It encapsulates the core functionalities and data management needed to control the player's interaction and movement within the game environment.

entity.monsters

Each class extends the “Monster" class, handling specific characteristics, image loading, and positioning logic for its corresponding monster type within the game environment.

i. Dragon.java

This class handles the spawning, image, and the statistics of the Dragon. It utilizes game data and resources to position the dragon in the game environment.

ii. Gnoll.java

This class manages the spawning, image, and the statistics of the Gnoll. As like other monster classes, it handles the positioning of the Gnoll within the game world.

iii. Goblin.java

Like other monster classes, this class manages the spawning, image, and statistics of the Goblin entity. It also handles the placement of Goblins within the game world and considering valid positions.

iv. Harpy.java

Like other monster classes, this class manages the spawning, image, and statistics of the Harpy character. It utilizes the game resources and information to position the Harpy in the game map.

v. Ogre.java

It manages the spawning, image, and statistical information of the Ogre. This class handles the placement and attributes of the Ogre entity within the game environment.

vi. Skeleton.java

Same as other monster classes, this class manages the spawning, image, and statistical attributes of the Skeleton. It also handles the positioning and attributes of Skeletons within the game map.

vii. Witch.java

Like other monster classes, this class manages the spawning, image, and statistical data of the witch. At the same time, it also handles the placement and attributes of witches within the game world.

game.io

i. Reader.java

This class is responsible for reading various game-related data from external files. It includes methods for retrieving information about majors, spells, monsters, and the game map from text files. The class parses specific sections of these files based on given identifiers, such as monster or major names, and extracts corresponding data for use within the game.

ii. SaveDB.java

This class is responsible for managing the game's save functionality through an SQLite database. It handles the saving and loading of game states, including player information such as name, position, chosen major, and credits, as well as details about defeated monsters. Additionally, this class manages database interactions, including saving player and monster data, retrieving saved game information, and deleting saved games when the save limit is reached.

game.ui

i. Map.java

This class is responsible for rendering and managing the game's map. It extends "JPanel" to handle graphical rendering and sets up the game screen with specific dimensions based on tiles. It initializes the player, handles keypresses, and updates the game state in a continuous loop. The class manages the player's position and drawing of the map, including obstacles.

ii. Print.java

This class is responsible for managing the display of textual information and story elements within the game. It handles various screens, including showing available majors, game progress, combat encounters, and storytelling aspects. The class interacts with the game's user interface to display text, ASCII art, headings, and provide information and choices to the player. It encapsulates functions for presenting game elements and controlling the flow of the game's narrative.

resource.ascii

In this file, it does not hold any classes, but contains all the ascii art used by the game.

- i. **ai_ascii.txt**
- ii. **book_ascii.txt**
- iii. **credit_ascii.txt**
- iv. **credits_ascii.txt**
- v. **csn_ascii.txt**
- vi. **dragon_ascii.txt**
- vii. **epilogue3n_ascii.txt**
- viii. **epilogue3y_ascii.txt**
- ix. **epilogue4_ascii.txt**
- x. **epilogue_ascii.txt**

xi. **gnoll_ascii.txt**
xii. **goblin_ascii.txt**
xiii. **harpy_ascii.txt**
xiv. **help_ascii.txt**
xv. **help1_ascii.txt**
xvi. **is_ascii.txt**
xvii. **load_ascii.txt**
xviii. **lose_ascii.txt**
xix. **menu_ascii.txt**
xx. **mmc_ascii.txt**
xxi. **ogre_ascii.txt**
xxii. **portal_ascii.txt**
xxiii. **se_ascii.txt**
xxiv. **skeleton_ascii.txt**
xxv. **story1_ascii.txt**
xxvi. **story2_ascii.txt**
xxvii. **story3_ascii.txt**
xxviii. **story4_ascii.txt**
xxix. **story_ascii.txt**
xxx. **win_ascii.txt**
xxxi. **witch_ascii.txt**

<u>resource.desc</u>

In this file, it also does not contain any classes, but all the information for map designing, archetypes, monsters, and the spells.

i. **majorsDesc.txt**
ii. **map.txt**
iii. **monstersDesc.txt**
iv. **SpellsDesc.txt**

resource.database

This file contains the database used for the save/load game functionality

i. **save.db**

resource.images

In this file, it contains all the images used in the map such as the monsters.

i. **player.png**
ii. **dragon.png**
iii. **gnoll.png**
iv. **goblin.png**
v. **harpy.png**
vi. **ogre.png**
vii. **skeleton.png**
viii. **tile.png**
ix. **Witch.png**

tile

i. Tile.java

This class serves as a representation of a tile within a game environment. It is responsible for managing the rendering of several types of tiles, including obstacles and different monster entities such as Goblin, Harpy, Skeleton, Witch, Ogre, Gnoll, and the Dragon. The class handles the drawing of these entities onto the game canvas based on the provided tile map. Additionally, it includes methods for loading tile images and rendering them onto the screen using Java's Graphics module.

ii. TileType.java

This class provides a foundational framework for storing an image ("BufferedImage") linked to a specific type of tile. It acts as a straightforward repository for image data and is used by other classes in the “tile" package to organize and retain tile images.

Extra Features:

1. Save game functionality

The database “save.db” is used to enhance the player experience, provide a way for player to record their current game status. It allows saving or loading the game functionality.

2. Abnormal input Handling

In the file “game.engine”, it contains the “CommandLineInputHandler” class that is responsible to handle the abnormal inputs. It does so by ignoring all other inputs except for specific choices.

3. Colourful Text

By enhancing the game to become more interesting and amazing, we provided the colourful text so that certain keywords be displayed on the screen are in distinct colours. This is implemented by way of html formatting. The Java Swing component we used had its content type set to “text/html” which allowed for highly specific text formatting.

4. Database Implementation

In the “SaveDB” class that mentioned above, the class manages the game’s save functionality using an SQLite Database, specifically the “save.db” file. It handles the database interactions, including saving player and monster data, retrieving saved game information, and deleting saved games when the save limit is reached.

5. ASCII Art

In the file named “resource.ascii”, it contains all the ASCII Art that will be used in the game. All the ASCII Arts are be stored in .txt files and the ASCII Art can make the game look more attractive and interesting.

Report Diagrams

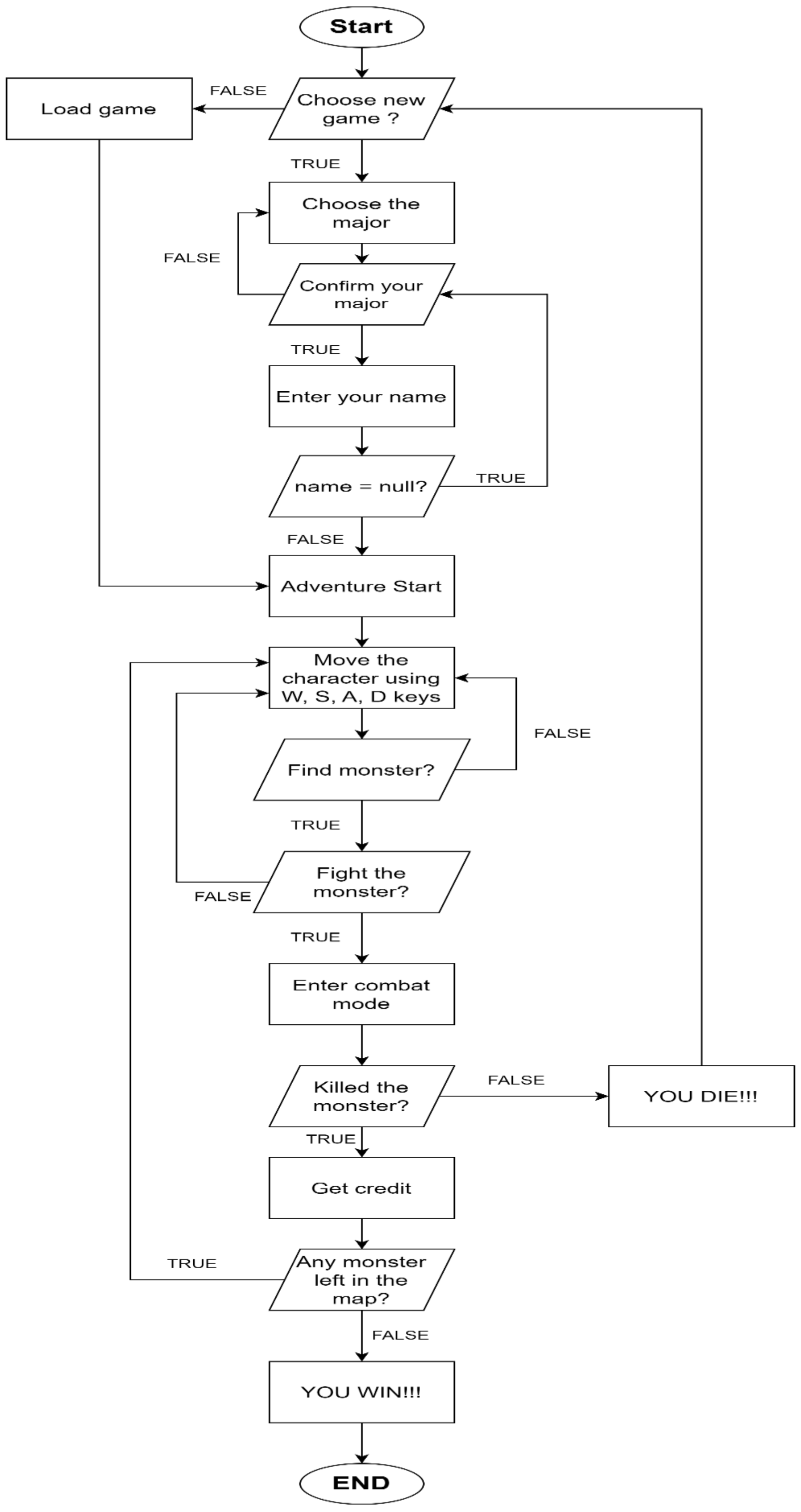


Diagram 1. Game Loop Diagram (Flowchart)

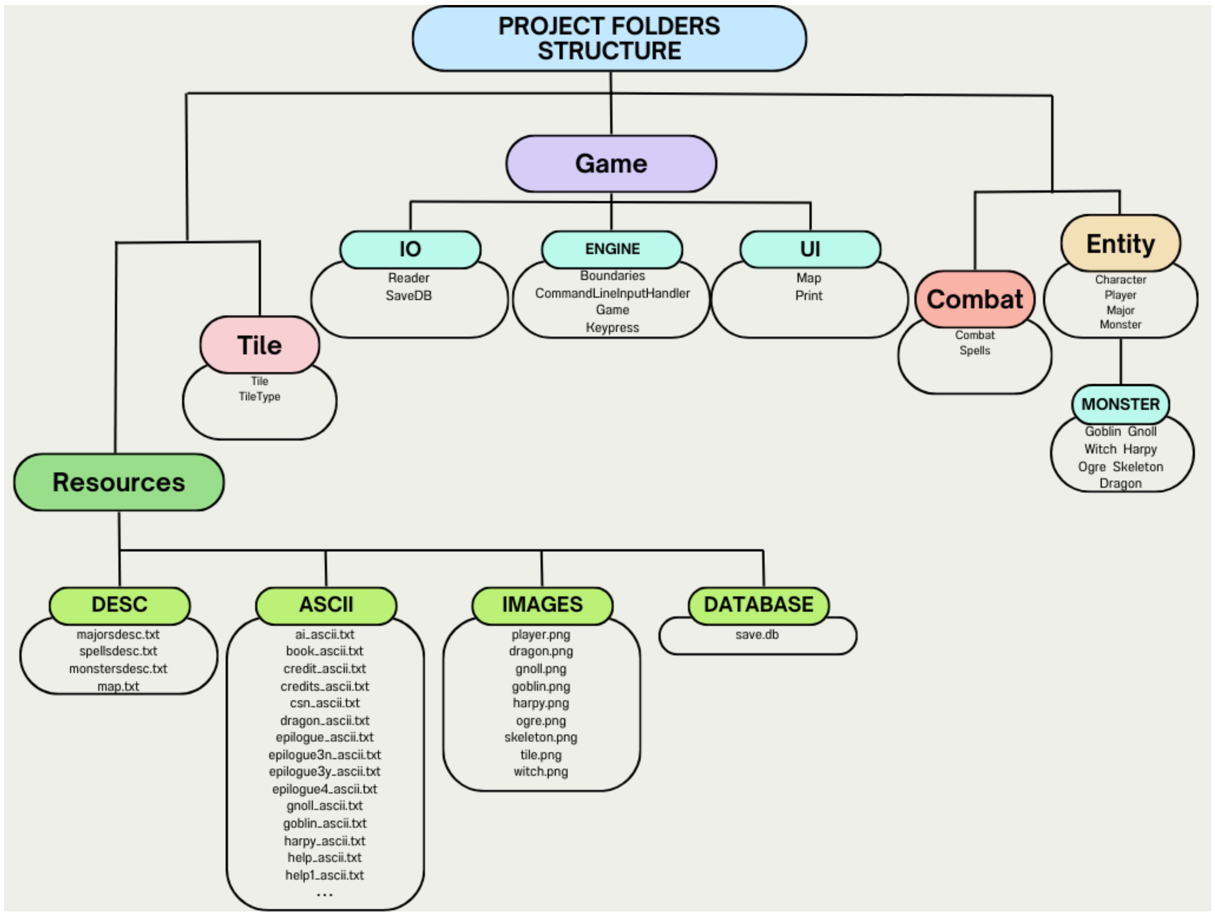
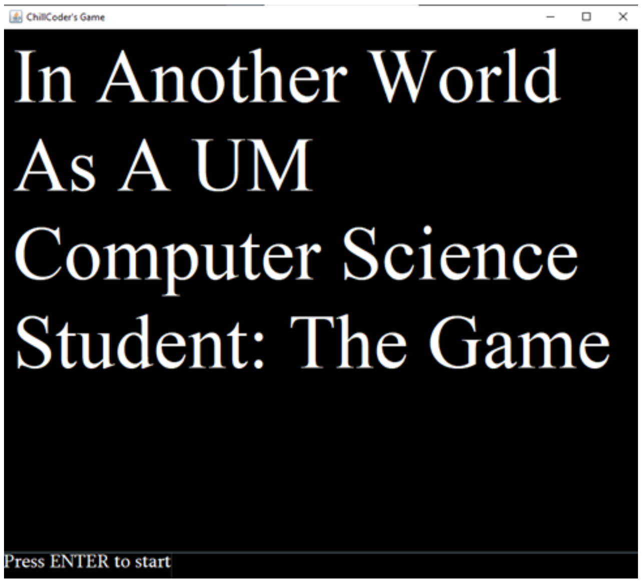


Diagram 2. Project Folders Structure

The first diagram describes the game loop. At the beginning of the game, the user must select a major and enter their name to start the adventure. Then the user can use the 'W', 'S', 'A', and 'D' keys to travel around the map. Inside the map, monsters will be generated randomly, and the user must combat the monsters to gain credits. During the battle, different spells can be cast but the user must obtain the required credits first to unlock the spells. To win the game, the user must successfully defeat all the monsters. If the user gets defeated by the monster, the game will restart from the beginning.

The second diagram shows the project folder structure. It is divided into three key parts that are "Game", "Entity and Combat", and "Resources and Tile". The game section consists of "IO", "Engine", and "UI". These store the reader, boundaries, game, keypress, map, etc. Then we have the four main entities of our game, which are "Character", "Player", "Major", and "Monster". The monster section has all the different monsters used in the game. To fight the monsters, there are combat and spells. Lastly, we have "Resources and Tile", which refers to the game resources including descriptions, images, ASCII arts, database, and tile.

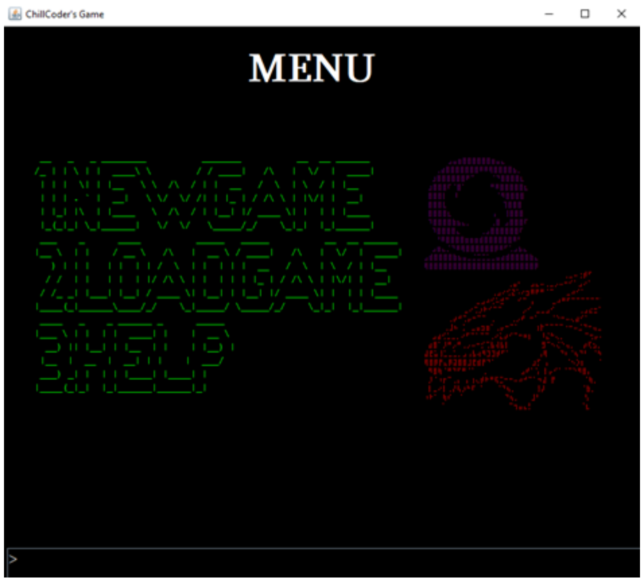
Sample Output


ChillCoder's Game

In Another World As A UM Computer Science Student: The Game

Press ENTER to start

Figure 1. Title Screen


ChillCoder's Game

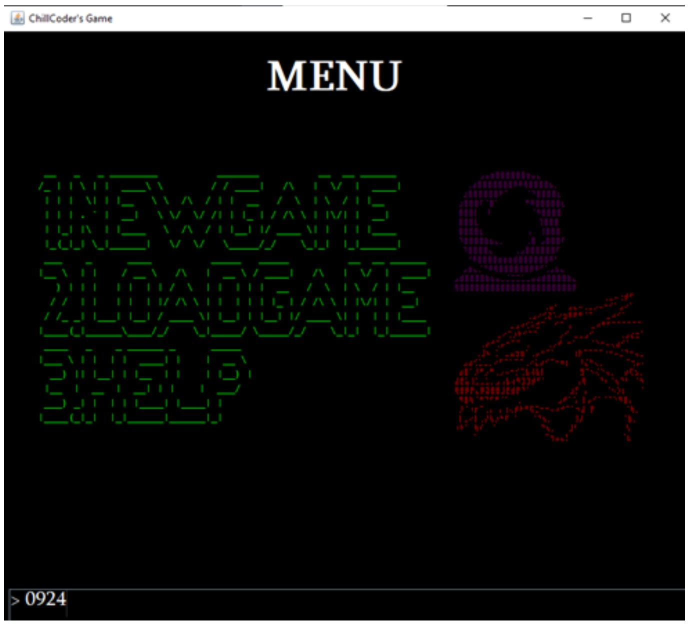
MENU

1.NEWGAME
2.LOADGAME
3.HELP

>

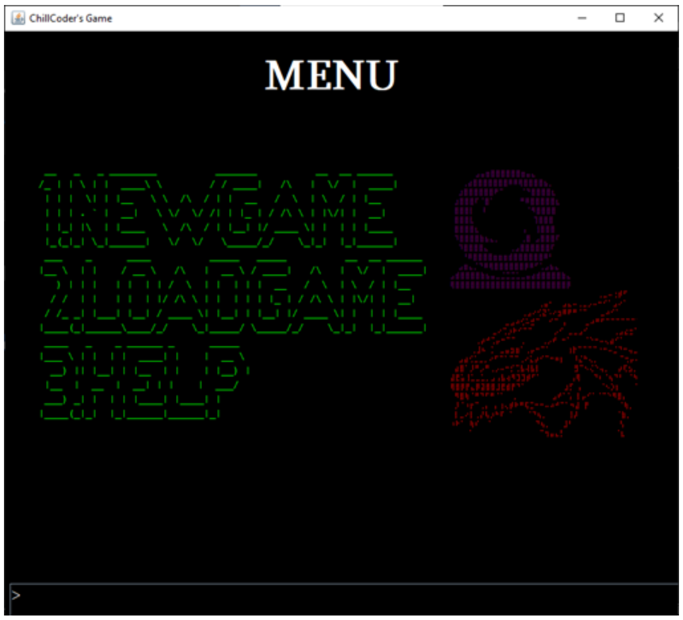
Figure 2. Menu Screen

The two images above show the starting screens when the program is run. The title screen displays the game's title and prompts the user to hit Enter to start the game. The menu screen shows the options for the user to choose from.



```
> 0924
```

Figure 3. User inputs invalid text



```
>
```

Figure 4. Result of Fig. 3

The images above highlight our implementation of abnormal input handling. Our program does not react to any irrelevant strings. All user input is retrieved as strings, as such, when the user inputs "0942" the String does not pass any of the conditions set for checking keywords. Additionally, the program cannot crash if the program asks for a number and the user inputs a string. This is because the only thing the user can input are strings.

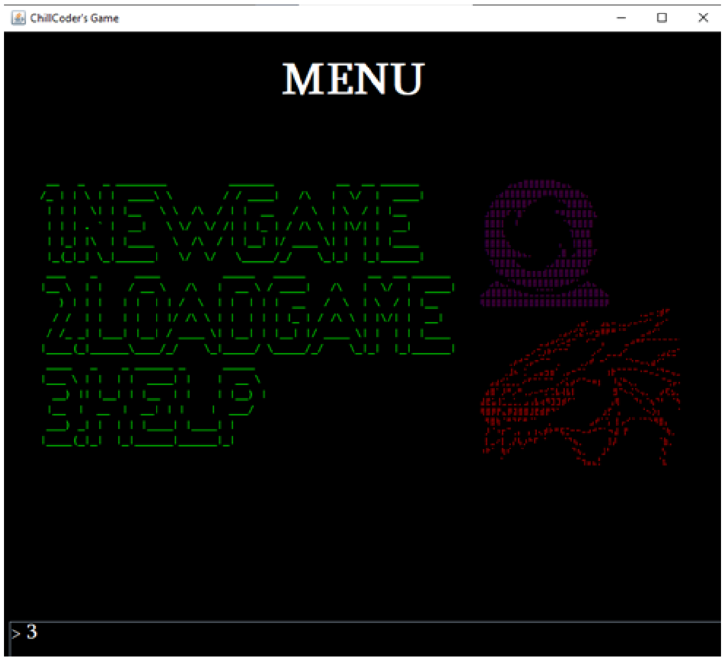


Figure 5a. User inputs valid text

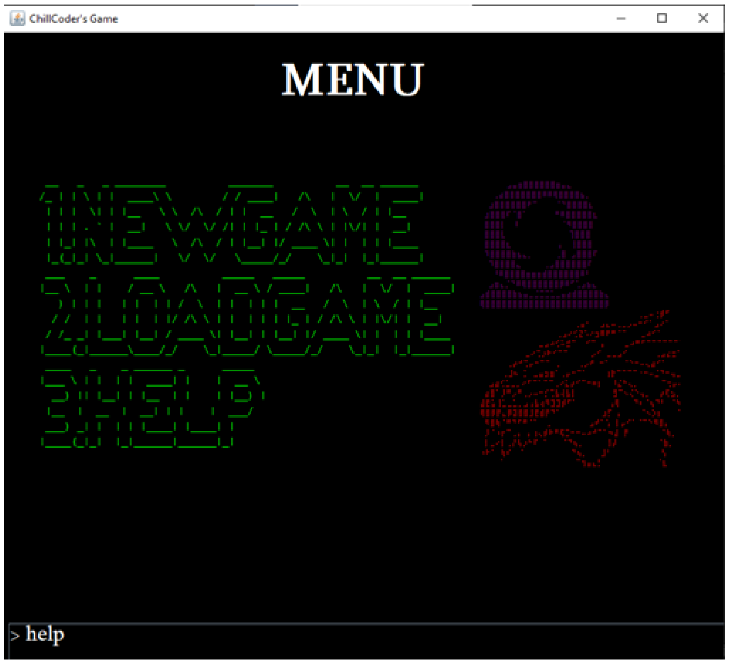


Figure 5b. User inputs valid text

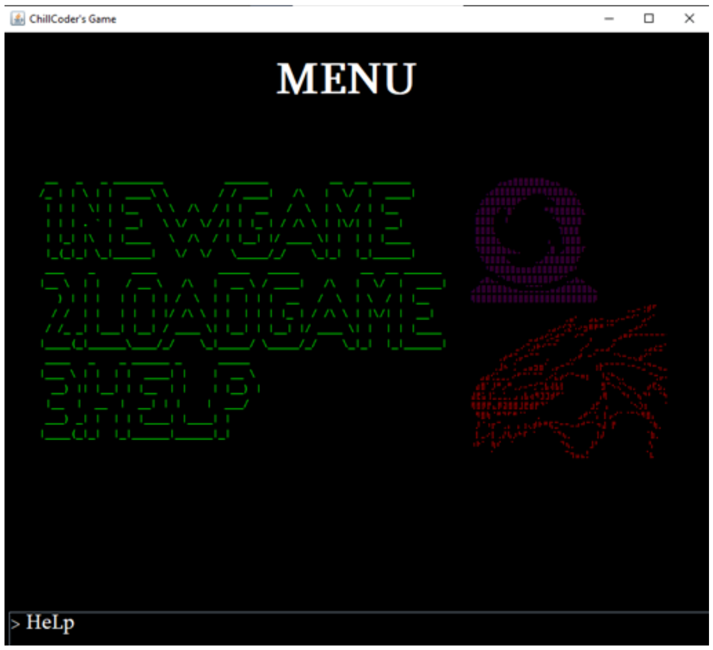
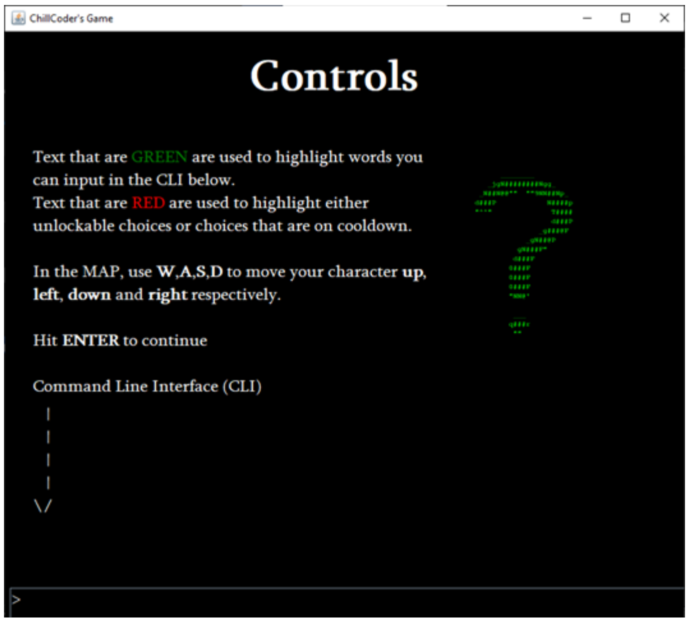


Figure 5c. User inputs valid text


ChillCoder's Game

Controls

Text that are GREEN are used to highlight words you can input in the CLI below.
Text that are RED are used to highlight either unlockable choices or choices that are on cooldown.

In the MAP, use **W,A,S,D** to move your character **up**, **left**, **down** and **right** respectively.

Hit **ENTER** to continue

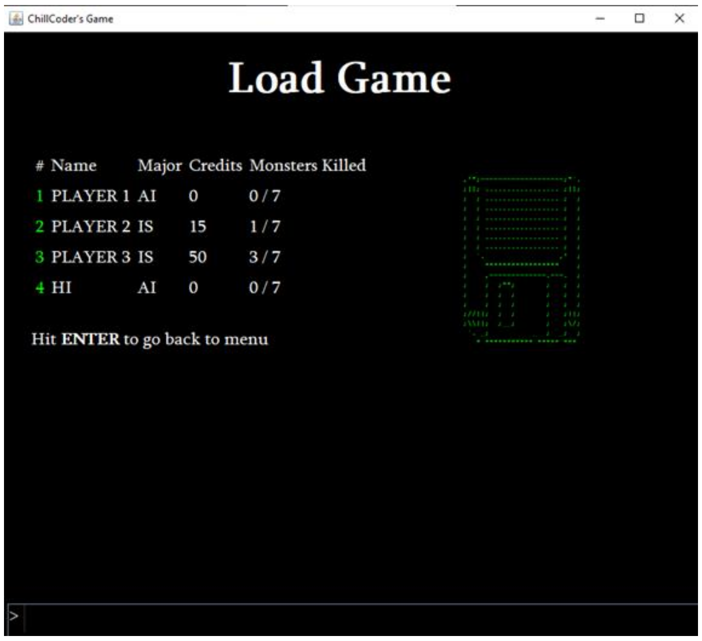
Command Line Interface (CLI)

```
 |
 |
 |
 |
\/
```

```
>
```

Figure 6. Result of Figures 5.

The figures 5 and 6 illustrates our versatile and functional input handling. In figures 5a, 5b, and 5c, the user inputs examples of valid texts that the program will appropriately react to. In figure 6, the program shows the user what they asked for, which was the help screen. Figure 6 only shows a portion of the help/tutorial screen which provides the user with a basic level of understanding regarding the game mechanics and functionalities.

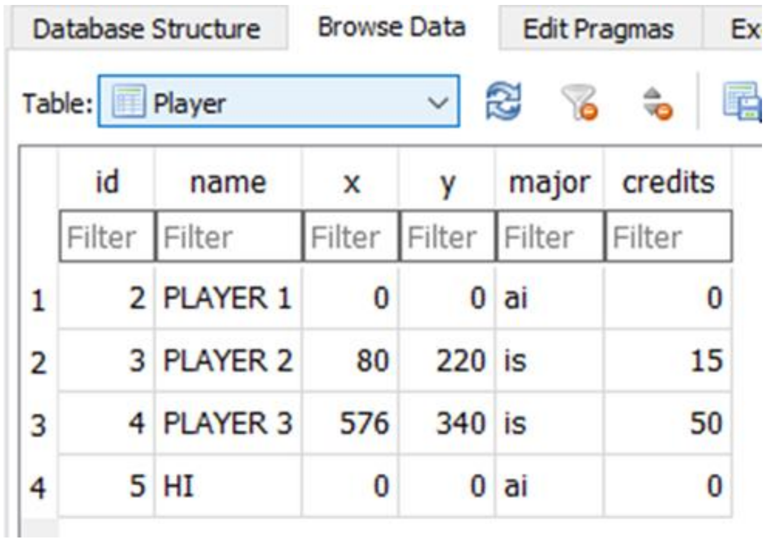


Load Game

```
# Name      Major Credits Monsters Killed
1 PLAYER 1  AI    0       0 / 7
2 PLAYER 2  IS    15      1 / 7
3 PLAYER 3  IS    50      3 / 7
4 HI        AI    0       0 / 7

Hit ENTER to go back to menu
```

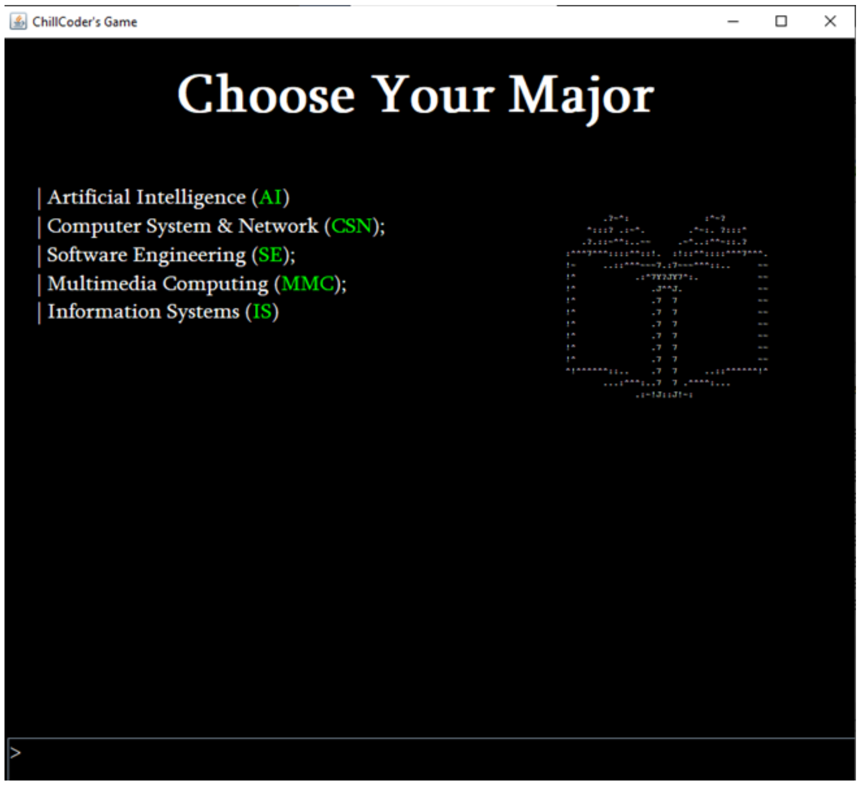
Figure 7. Load Game Screen



	id	name	x	y	major	credits
	Filter	Filter	Filter	Filter	Filter	Filter
1	2	PLAYER 1	0	0	ai	0
2	3	PLAYER 2	80	220	is	15
3	4	PLAYER 3	576	340	is	50
4	5	HI	0	0	ai	0

Figure 8. DB Browser screenshot of save.db Player Table

Figure 7 presents the screen showed to players when opting to play with an existing save. The load game functionality depends on the “save.db” file which is a database file that contains tables that hold the information for each save. Figure 8 shows the Player Table from the save.db file. There are other tables for each monster of the game and the database is managed by use of SQLite.

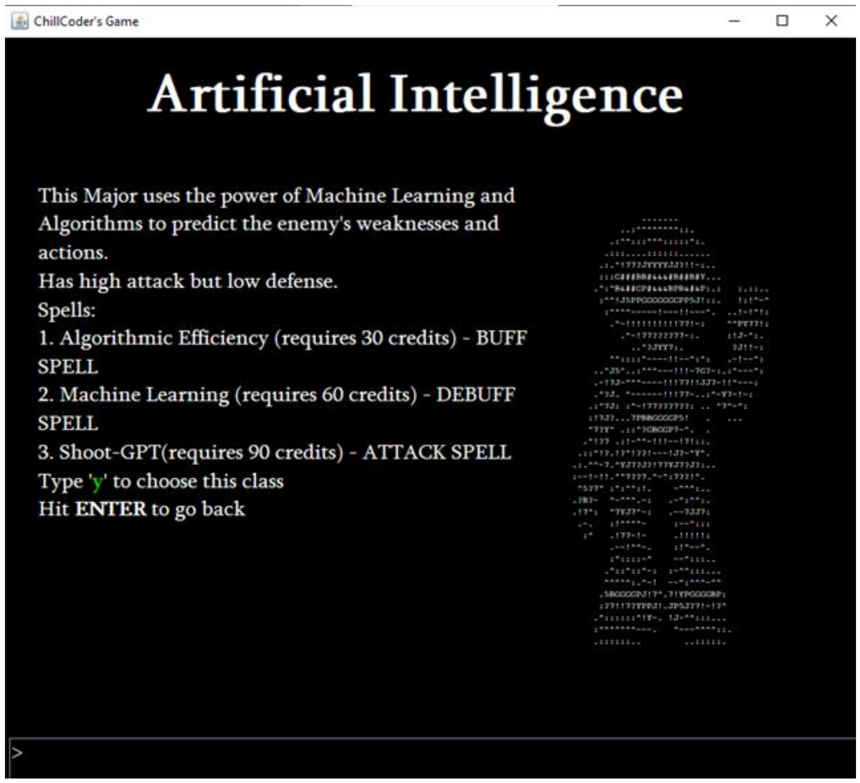

ChillCoder's Game

Choose Your Major

| Artificial Intelligence (AI)
| Computer System & Network (CSN);
| Software Engineering (SE);
| Multimedia Computing (MMC);
| Information Systems (IS)

>

Figure 9a. New Game Screen – Choose Archetype

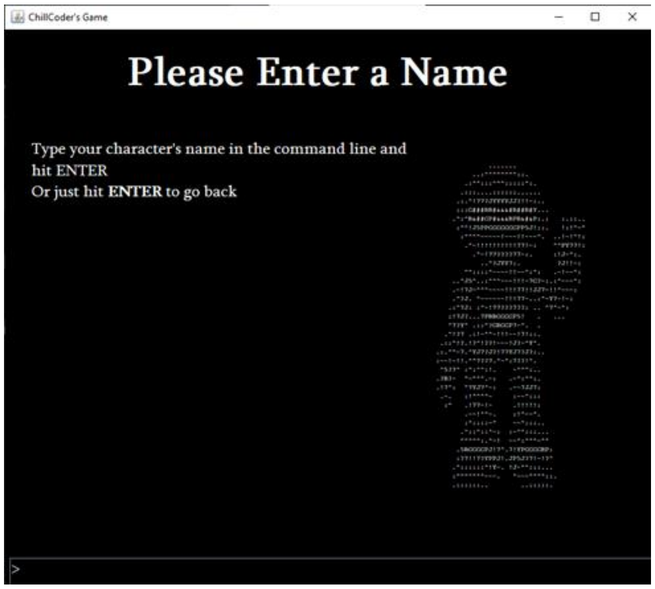

ChillCoder's Game

Artificial Intelligence

This Major uses the power of Machine Learning and Algorithms to predict the enemy's weaknesses and actions.
Has high attack but low defense.
Spells:
1. Algorithmic Efficiency (requires 30 credits) - BUFF SPELL
2. Machine Learning (requires 60 credits) - DEBUFF SPELL
3. Shoot-GPT(requires 90 credits) - ATTACK SPELL
Type 'y' to choose this class
Hit **ENTER** to go back

>

Figure 9b. New Game Screen – Confirm Archetype


ChillCoder's Game

Please Enter a Name

Type your character's name in the command line and
hit ENTER
Or just hit **ENTER** to go back

>

Figure 9c. New Game Screen – Choose Name

Figures 9a, 9b, and 9c show the process of starting a new game. The user is prompted to choose an archetype (major) and a name for your player character.

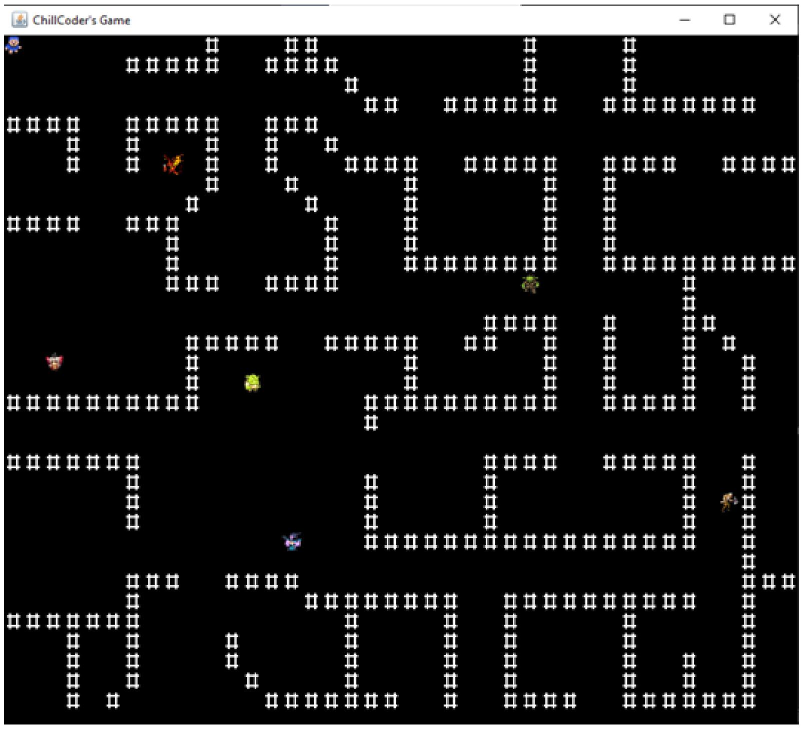


Figure 10a. Map Example

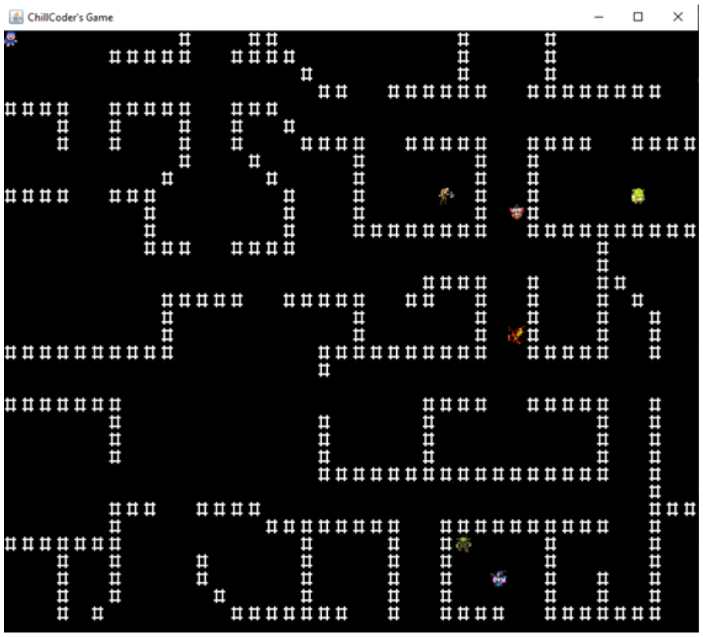
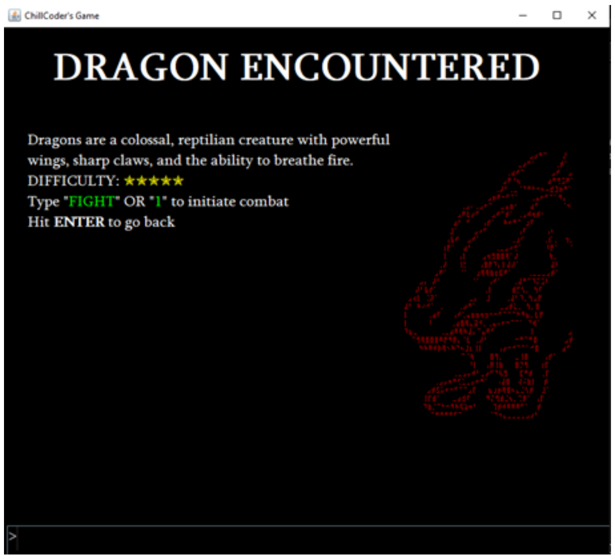


Figure 10b. Map Example

Figure 10 shows two examples of the random monster generation. The map design of all iterations of the game is the same, however the monsters can spawn anywhere on the map if it is an empty space. This gives the player a sense of dynamism and helps make the game feel less stale.

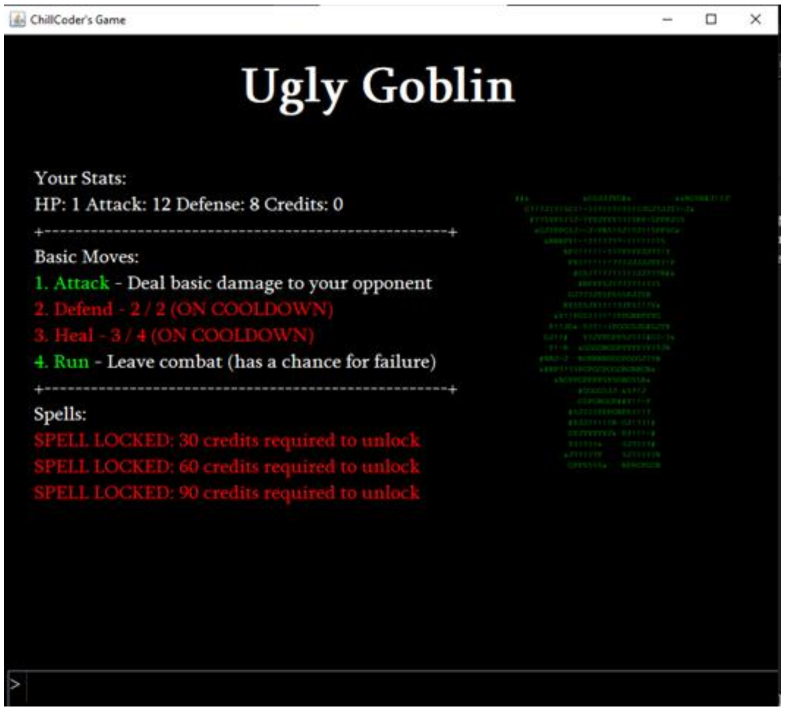


DRAGON ENCOUNTERED

Dragons are a colossal, reptilian creature with powerful wings, sharp claws, and the ability to breathe fire.
DIFFICULTY: ★★★★★
Type "FIGHT" OR "1" to initiate combat
Hit ENTER to go back

```
>
```

Figure 11a. Combat Screen



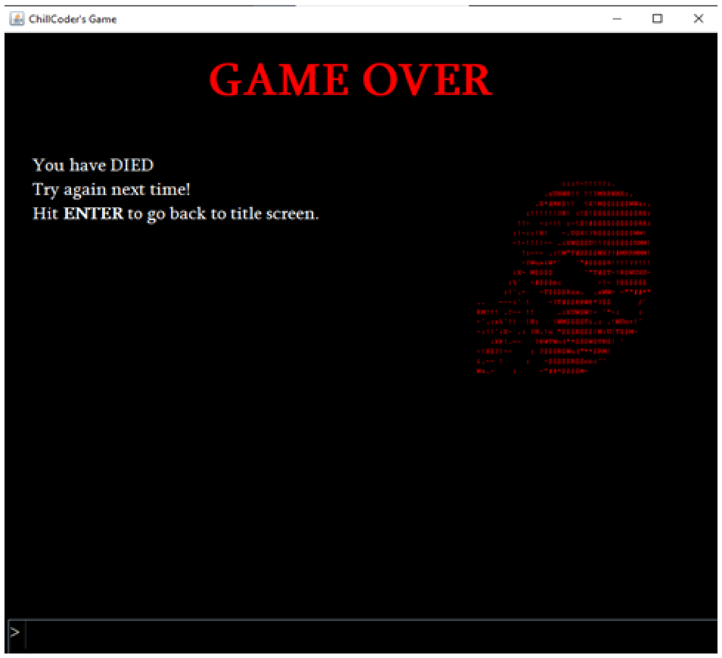
```
Ugly Goblin

Your Stats:
HP: 1 Attack: 12 Defense: 8 Credits: 0
+--------------------------------------------------+
Basic Moves:
1. Attack - Deal basic damage to your opponent
2. Defend - 2 / 2 (ON COOLDOWN)
3. Heal - 3 / 4 (ON COOLDOWN)
4. Run - Leave combat (has a chance for failure)
+--------------------------------------------------+
Spells:
SPELL LOCKED: 30 credits required to unlock
SPELL LOCKED: 60 credits required to unlock
SPELL LOCKED: 90 credits required to unlock

>
```

Figure 11b. Combat Screen

Figure 11a and 11b shows some examples of what the user might see when engaging in combat with a monster.

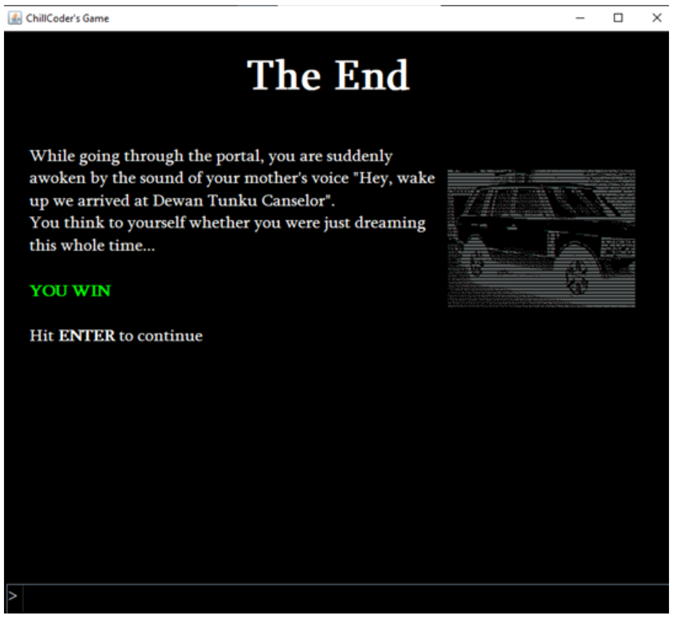


```
GAME OVER

You have DIED
Try again next time!
Hit ENTER to go back to title screen.

>
```

Figure 12 Game Over



The End

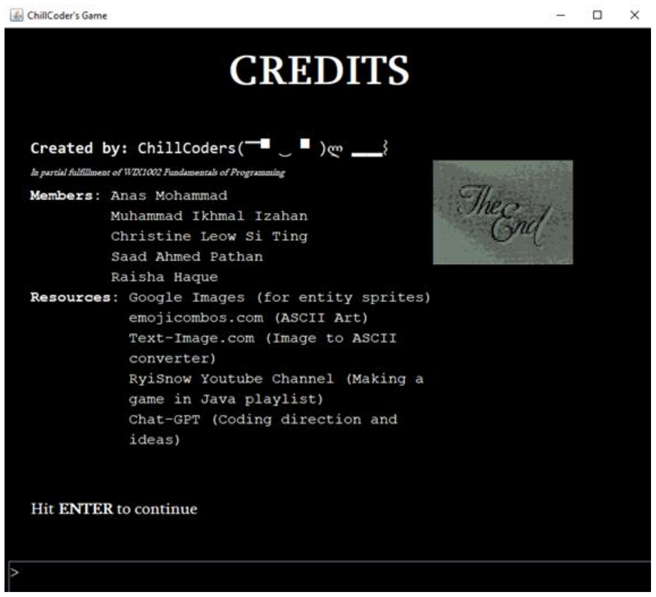
While going through the portal, you are suddenly awoken by the sound of your mother's voice "Hey, wake up we arrived at Dewan Tunku Canselor".
You think to yourself whether you were just dreaming this whole time...

YOU WIN

Hit **ENTER** to continue

>

Figure 13 Game Win



CREDITS

```
Created by: ChillCoders(̿▀̿ ‿ ̿▀̿ ̿)ლ ___}
In partial fulfillment of WIX1002 Fundamentals of Programming
Members: Anas Mohammad
         Muhammad Ikhmal Izahan
         Christine Leow Si Ting
         Saad Ahmed Pathan
         Raisha Haque
Resources: Google Images (for entity sprites)
           emojicombos.com (ASCII Art)
           Text-Image.com (Image to ASCII
           converter)
           RyiSnow Youtube Channel (Making a
           game in Java playlist)
           Chat-GPT (Coding direction and
           ideas)
```

Hit **ENTER** to continue

>

Figure 14. End Credits

For the 3 images above, these are what the player sees when they finish the game, either successfully or unsuccessfully.