CS 319

Object-Oriented Software Engineering

Project Final Report

Defender

GROUP 2-D

Taha Khurram - 21701824

Mian Usman Naeem Kakakhel - 21701015

Muhammad Saboor - 21701034

Sayed Abdullah Qutb - 21701024

Balaj Saleem - 21701041

1. Introduction

For the implementation of the game Defender, we decided to use C++ and SDL2 library in the design report. We created a GitHub repository [1] for the game and every one used this repository to share the code with other members of the group. We chose Linux as the OS environment, because we do not need any extra tools and packages to start game development.

In the first iteration, a basic window frame was made that included the main game loop, key listener and the required display manager to render in-game components so that other components of the game can easily be tested. Next, a point system was created to catch a point on the map and translate it for the display view. After this the spaceship was implemented however, at that time the spaceship only supported basic movement and the ability to shoot bluebolt bullets. In the first implementation we were also able to integrate the asteroid and the enemy model classes were made.

In the second iteration, we applied the strategy design pattern to the enemy fire behavior. Threading was used to separate backend processing and front-end processing inorder, the display manager was also changed according to the design report (second iteration). Next, the missile and nuclear bomb were integrated with the spaceship allowing the use of both weapons in game. The enemy, human and warpzone classes were also integrated into the game and the collisions among components were detected and handled. The HUD with scoring system and the map view were implemented, moreover the datamanager was implemented inorder to store and load the high score. Next, the mouse listener was implemented and integrated following which the main menu and the shop frames (with buttons) were integrated. However, the purchase skin option from the shop menu was not implemented due to time constraints. The explosion effects were also added to appear at the death of any enemy unit and in the end the sound effects were implemented and added.

2. Design Changes

Our implemented design has very few changes from the design we proposed [2] in the second iteration.

- Due to performance issues in the game, instead of loading the sprite texture in every game loop, the textures of each displayable object was created at the time of its instantiation and the texture was stored as an attribute in the class. Thus, the time taken to render the objects on the screen was reduced considerably.
- Explosion class was added to the design to include explosion effects whenever any bullet or enemy is destroyed.
- Removed mutate() from Enemy class, as an enemy object can not destroy and then recreate itself. This functionality was implemented in the GameFrame class instead.
- Removed orientation attribute from the Human class, as everything was handled by humanState attribute. Furthermore, updatePosition() was added and implemented in Human class to handle the movement of a human.
- Removed description attribute from Item class as it was not needed for the functionality of Item.

- Relevant attributes were added in the SpaceShip, Enemy, Explosion, and Button classes for adding sound effects in the game.

3. Lessons Learnt

We learnt the following lessons during the course of the project.

- We learnt that during runtime, if a certain object is required again, it should not be created again and again. It should be created for the first time and then kept for reuse. For example, when rendering sprites to the screen, the sprite is first loaded into a texture and then that texture is copied on to the renderer. Loading the sprite into a texture takes the most time of the process. Thus, keeping the texture after loading it once increases the game performance substantially.
- If a single instance of an object is repeatedly required by a lot of classes and it is not changed throughout the execution of the program, then it is a good idea to keep that object as a static global object. This reduces the time and space required when passing the object through the parameters. For example, in our game SDL_Renderer object is required by all displayable classes and it is not changed throughout the game [1]. So we provided it as a static global object instead of passing it through the parameters.
- Whenever we are using threading, if multiple threads are supposed to access the same memory location, then it should be made sure that they are not accessing the memory location at the same time. For example, in our game, backend processing is separated from the frontend processing using threads [1]. Both threads access the position of all the enemies. Backend thread accesses the positions while updating them whereas frontend thread accesses the positions while displaying them on the screen. SDL provides a semaphore which handles whether a certain thread is accessing a memory location or not and makes sure that other threads wait for a thread to finish its processing on that memory location.
- The quality of analysis and design report is extremely important because they come in handy during the implementation process if they were prepared carefully. For example, during the implementation of our game, when creating the Enemy class and its children, the analysis report was required to implement the updatePosition() of each Enemy [1]. Since we covered their movement behaviors in our Analysis document, it was very helpful during our implementation.

4. User's Guide

4.1 Introduction

In the Defender, the Earth is being invaded by aliens and you, as the player, will commandeer a spaceship in an attempt to shoot down all enemy aliens. The player aims to avoid taking damage from the enemy aliens whilst trying to stop them from abducting humans on the surface of Earth. The aliens are abducting humans so that they can absorb their spiritual energy and mutate resulting in a massive boost in power. The aliens aim to capture and kill all humans on the planet as the total spiritual power absorbed will be enough for their entire species to mutate and unlock their full potential becoming extremely powerful with the intention to now destroy the Earth as there are no more surviving humans. It is solely up to the player to save the world and his species from total annihilation [3].

The defender is an endless runner game that allows the player to control a spaceship and shoot down invading aliens.The player will have to conserve their ammo and prioritize saving the captured humans to prevent the aliens from becoming more powerful and consequently destroying the player's home world.

4.2 System Requirement

Display resolution : 1080p

Processor: Core i3 or above

RAM: 2GB or more

Keyboard and Mouse

Operating System: Linux (Debian Recommended as the game is tested on Debian)

(Note: install.sh file should be used to install required libraries before running the game).

4.3 How to Play

Rules:
The player will be controlling a spaceship and has to shoot down enemies, if the players health or fuel amount reaches 0 the game will be over and the player will earn score for the duration they stay alive. If the enemies are able to deduct humans they will mutate into stronger enemies (crescent) and if the enemies are able to abduct all humans, all the enemies will undergo mutation and turn into crescent and any new enemy generated will also be a crescent making it harder for the player to survive.

Main menu:

As soon as the player enters the game they will be taken to the main menu where the player can select "New Game" or "Quit".

1) By selecting the Quit option, the game will close.
2) If the player selects the New Game option, the player will be taken to the game screen.

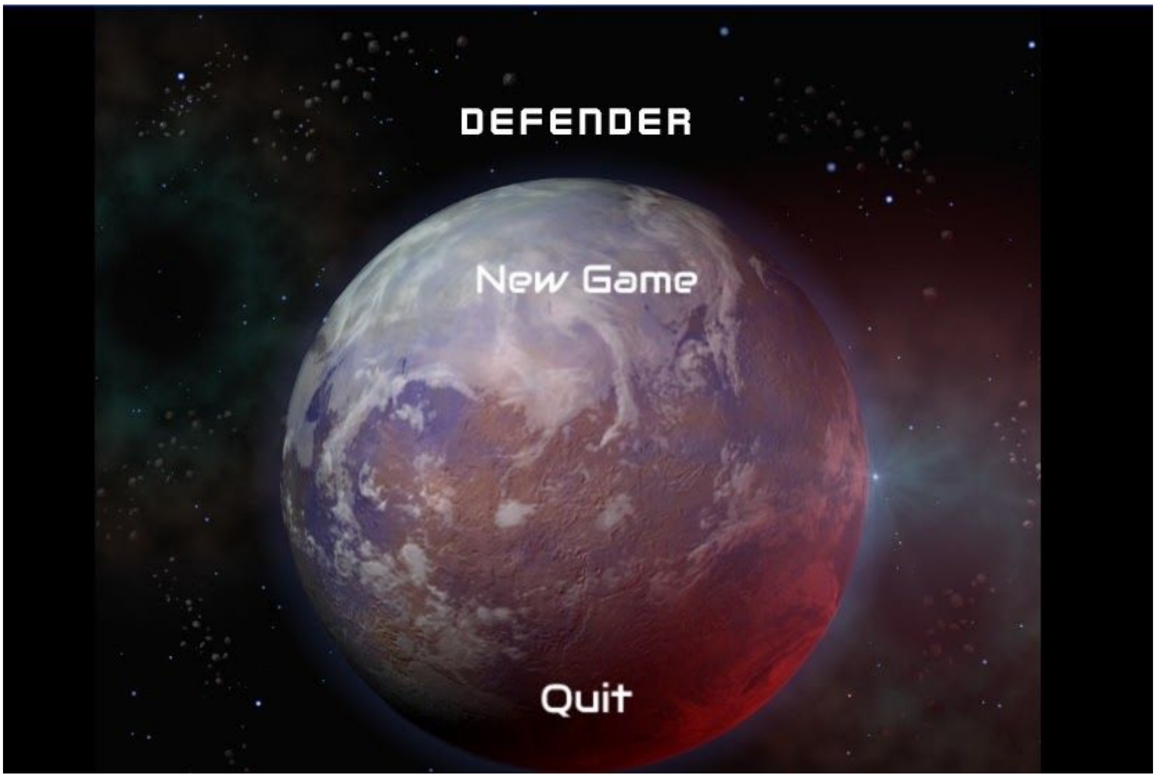


Figure 1 - Main menu

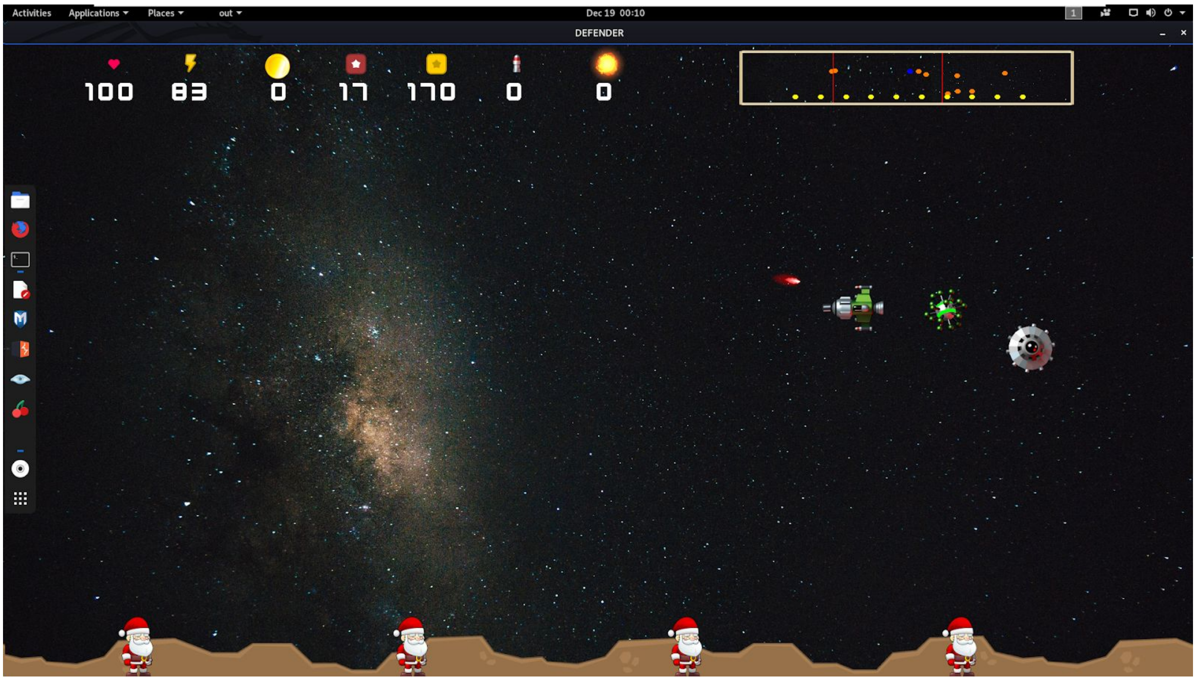


Figure 2 - Game screen

Spaceship:

The player will be incontrol of the spaceship and will get to use bluebolt bullets, missiles and nuclear bombs as weapons. The spaceship will have a health and fuel amount and if either of those reach 0 the game will be over.

Health:

The heart ❤ in the figure shows the amount of health left, if health reaches 0 the spaceship is destroyed and results in game over. The player can refill this amount by catching the health pickup.

Fuel:

The fuel ⚡ shows the amount of fuel left, every second the player loses 2 fuel and if the fuel amount reaches 0 the spaceship will crash resulting in a game over. The player can refill this amount by catching the fuel pickup.

Coins:

The coin 🪙 shows the amount of coins the player has; each time the player kills an enemy they get 5 coins. The player can later spend these coins in the shop to purchase upgrades.

Score:

The score ★ shows the current score of the player, the player increases their score by staying alive and the score is updated every second.

High score:

The high score shows the highest score reached by the player so far. If the player is playing the game for the first time the high score will be 0 and whenever the player's current score is greater than the high score it gets updated in real time.

Missile:

The missile is a type of weapon the player can shoot by pressing the Z key. The missile chases the closest enemy and upon collision the enemy will be killed. However, the player cannot spam the missile as the missile has a cooldown of 3 seconds.

Nuclear bomb:

The nuclear bomb is a type of weapon the player can use by pressing the C key. The nuclear bomb when used kills all the enemies on screen. The player has to wait for 10 seconds to reuse this weapon.

Map:

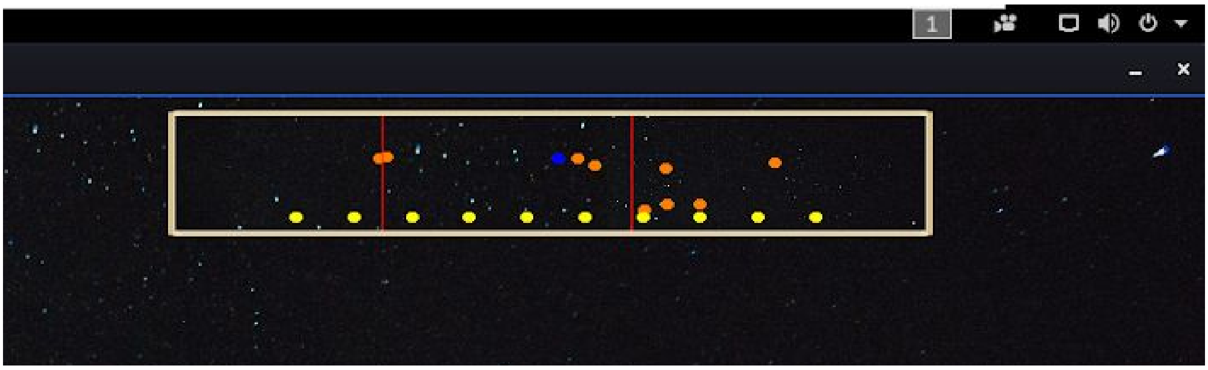
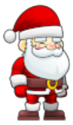


Figure 3 - Map

The map shows the position of the spaceship (The blue dot),the position of all the enemies (orange dot) and humans (yellow dot). The red bars are used to show which section of the overall map the spaceship is in.

Human:



The human is initially located on the ground and the bugeye enemy will try to abduct the humans and take them to their spaceship so it is up to the player to save them.

The pause menu:

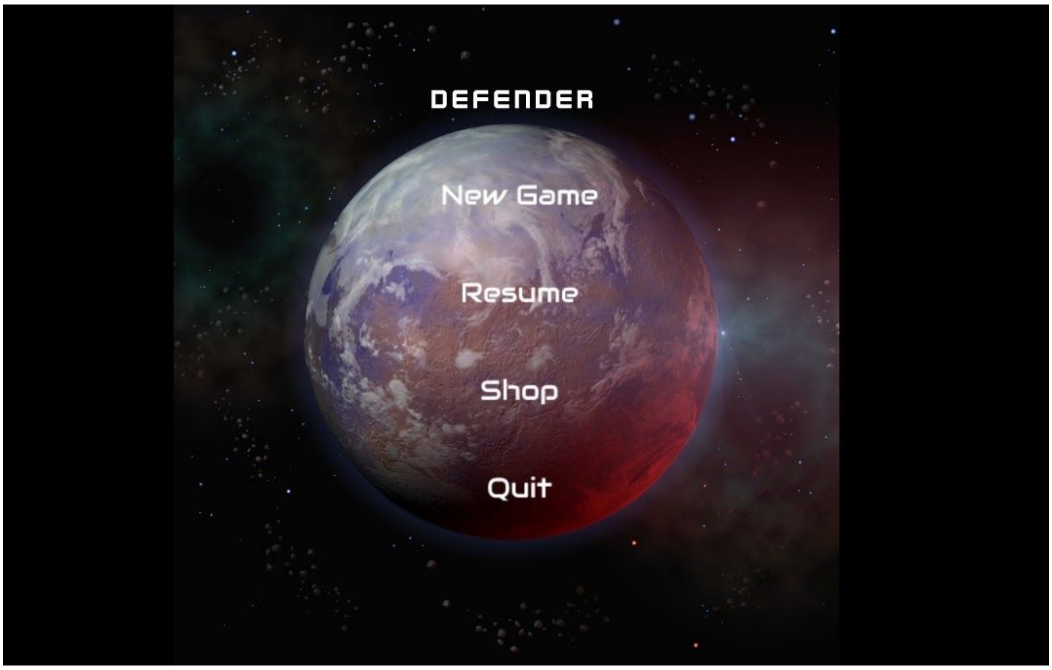


Figure 4 - Pause menu

When the player presses the esc key, they will be taken to the pause menu.

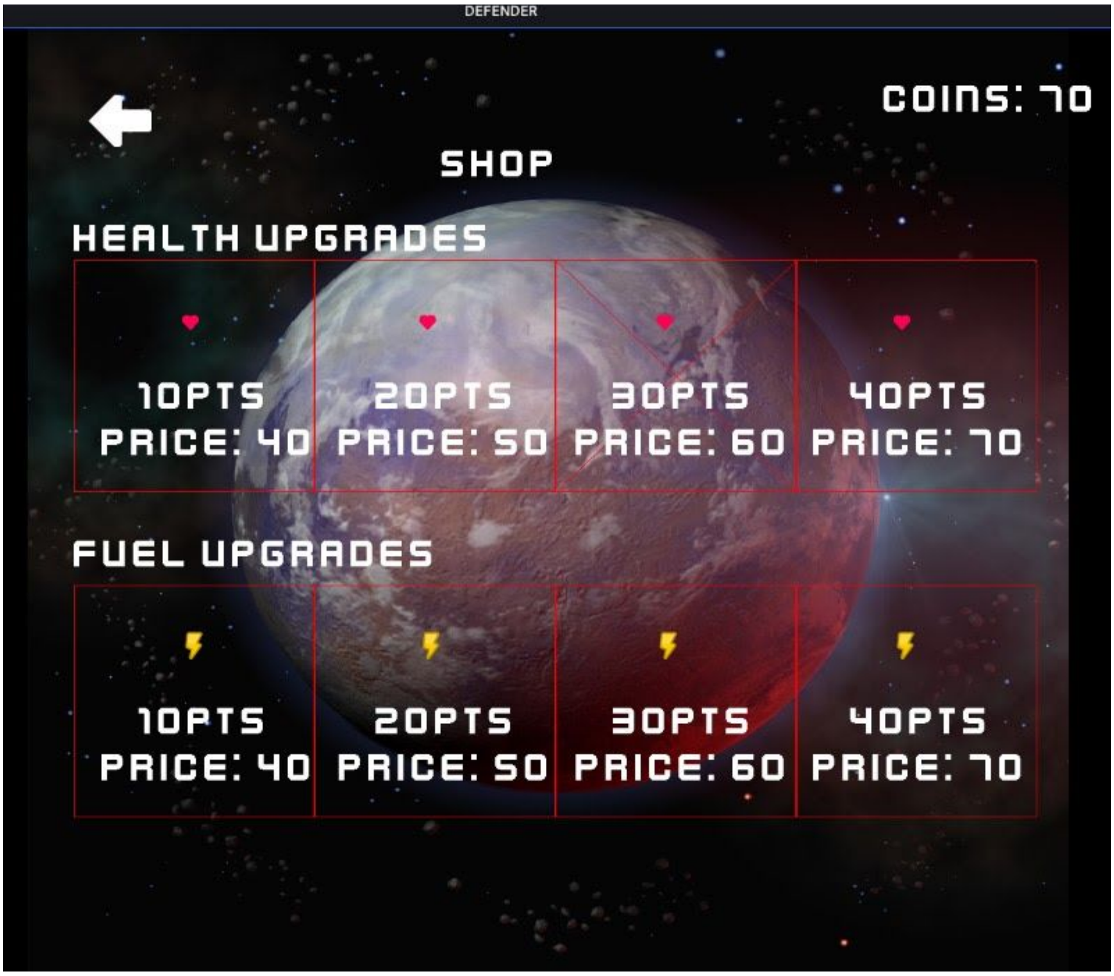
1) If the player presses the resume button, they will be taken back to the game.
2) If the player presses the new game button, the old game finishes and the player gets to start a new game
3) If the player presses the shop button, they will be taken to the shop menu where they can buy upgrades.

The shop menu:

Inside the shop menu the player has the option to purchase upgrades for their spaceship. However the purchases made in one game do not carry on to the next game!

1) The player can make health upgrades by increasing the maximum amount of health.
2) The player can make fuel upgrades by increasing the maximum amount of fuel.

The upgrades will stack, for example: if the player buys 10 PTS of health and then 20 PTS of health they will have a total of +30PTS of health. Moreover, the player cannot make the same purchase more than once as items already bought will be crossed.



HEALTH UPGRADES

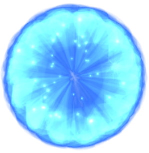
♥ 10PTS PRICE: 40	♥ 20PTS PRICE: 50	♥ 30PTS PRICE: 60	♥ 40PTS PRICE: 70

FUEL UPGRADES

⚡ 10PTS PRICE: 40	⚡ 20PTS PRICE: 50	⚡ 30PTS PRICE: 60	⚡ 40PTS PRICE: 70

Figure 5 - Shop menu

Warp Zone:



There will be random warp zones generated on the map that will transport the spaceship(player) from one location to another. The warp zone will be generated randomly on the current screen.

Asteroid:

The spaceship will have to dodge falling asteroid which on collision will deal 50 damage to the player’s spaceship.

Enemies:

BugEye: These enemies are responsible for picking up humans. It will not fire any weapon towards the spaceship. Its health will be 100 points. Its speed will be 3. It will find the nearest human and go towards it to abduct it. After abducting, BugEye will start moving upwards and when it reaches the top, it will mutate itself into Crescent.

Crescent: This enemy results after the mutation of weaker enemies and will only attack the player. Its health will be 100 points. Its speed will be 5. It will be equipped with OrangeGlow. This enemy will move towards the spaceship and start orbiting around it while shooting at the spaceship.

Saucer: This is a low-level enemy which shoots RedBolt at the player. Its health will be 50 points. Its speed will be 3. The direction of this enemy is randomly decided at generation time and then it continues to move in that direction until it bounces off of the edges of the map and continues in its new bounced direction.

Saucer Blades: These are stronger versions of Saucers in terms of speed. The speed of Saucer Blades will be 4. Its health, damage and bullets will be the same as Saucer. The direction of this enemy is randomly decided at generation time and then it continues to move in that direction until it bounces off of the edges of the map and continues in its new bounced direction.

Slicer: This is a low-level enemy that only attacks the player. Its speed is 3. Its health will be 50 points. It will be equipped with RedBolt. The direction of this enemy is randomly decided at generation time and then it continues to move in that direction until it bounces off of the edges of the map and continues in its new bounced direction.

Spikey: These enemies will try to attack the player by getting close to the spaceship and exploding dealing damage of 15 points of damage and its speed will be 3.

Health pickup:

The player will be able to utilize a health pickup which will restore the space ship’s health to full, this pickup only generates when the player’s health is below 30. It has the same icon as Health ♥ .

Fuel pickup:

The player will be able to utilize a fuel pickup which will restore the space ship’s fuel to full. this pickup only generates when the player’s fuel is below 30. It has the same icon as fuel ⚡ .

Weapons:

1) **Redbolt :** These are the bullets fired by all enemies except the crescent and they deal 20 damage to the spaceship.
2) **OrangeGlow :** These bullets are fired only by crescent enemies and deal 30 damage to the spaceship.
3) **BlueBolt :** This is the bullet fired by the spaceship and deals 100 damage to the enemy.
4) **Missile:** The spaceship will fire missiles that chase the closest enemy and kill them.
5) **Nuclear bomb:** The spaceship can fire a nuclear bomb which destroys all enemies on screen.

Controls:

In the game screen the player will be controlling the spaceship using the directional arrow keys and the z, x and c keys.

UP arrow key: by pressing the up arrow key the spaceship will move upwards

DOWN arrow key: by pressing the down arrow key the spaceship will move downwards

RIGHT arrow key: by pressing the right arrow key the spaceship will move to the right.

LEFT arrow key: by pressing the left arrow key button the spaceship will move to the right.

ESC key: by pressing the esc key the player will be taken to the pause menu.

Z key: by pressing the Z key the spaceship will shoot the bluebolt bullet.

X key: by pressing the X key the spaceship will shoot the missile.

C key: by pressing the C key the spaceship will use the nuclear bomb and kill all enemies on screen.

Both the missile and nuclear bomb weapons have cooldowns, the missile has a 3 second cooldown and the nuclear bomb has a 10 second cooldown.

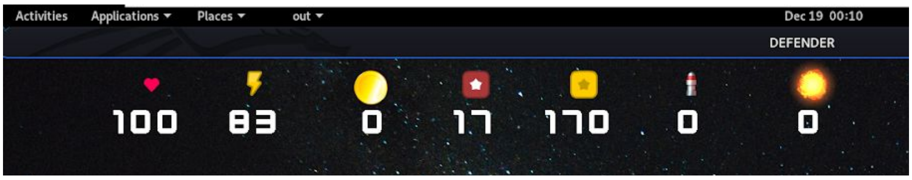


Figure 5 - a cooldown of 0 seconds remaining on both the missile and nuclear bomb

5. Build Instructions

The build instructions for Linux system are as follows:
(All given commands need to be run in the linux terminal)

1. Clone the following GitHub repository using the following command:
   git clone https://github.com/usman-kakakhel/CS319-2D-DE.git
2. Install the related SDL2 libraries used in the game using the following commands or simply run install.sh using bash (*bash install.sh*):
   sudo apt install libsdl2-dev libsdl2-image-dev libsdl2-ttf-dev libsdl2-mixer-dev
3. Go to the src folder in the cloned repository's folder and compile the source code using the make command:
   cd CS319-2D-DE/src && make

6. Credits

6.1 Mian Usman Naeem Kakakhel

- 2 State Diagrams, 1 Activity Diagram, and Class Diagram in Analysis Report Iteration 1

- Sub-system Decomposition, Class Diagram, and trade-offs in Design Report Iteration 1.
- 2 Activity Diagrams and Class Diagram in Analysis Report Iteration 2.
- Purpose of the system, Design goals, sub-system decomposition, class diagram in Design Report Iteration 2.
- Design changes and Build Instructions in Final Report Iteration 2.
- Defender, GameFrame, MenuFrame, DisplayManager, EventListener, Camera classes in implementation.

6.2 Muhammad Saboor

- 2 Activity Diagrams and Class Diagram in Analysis Report Iteration 1
- Sub-system Decomposition, Class Diagram, and design patterns in Design Report Iteration 1.
- 3 State Diagrams and UI Mockups in Analysis Report Iteration 2.
- Failure boundary condition, sub-system Decomposition, Class Diagram, and design patterns Iteration 2.
- Introduction and Lessons learnt in Final Report Iteration 2.
- ShopFrame, MassDestructionWeapon, NuclearBomb, DataManager, MapView, and SpaceShip classes in implementation.

6.3Taha Khurram

- Introduction, Overview, Functional Requirements in Analysis Report Iteration 1
- Sub-system decompositions' explanation in Design Report Iteration 1
- Introduction, Overview, and Improvement Summary in Analysis Report Iteration 2
- Sub-system decompositions' explanation, Improvement summary, and trade-offs in Design Report Iteration 2
- User guide and Glossary in Final Report Iteration 2.
- Weapon, TargetedWeapon, Missile, Bullet, RedBolt, BlueBolt, OrangeGlow, Asteroid classes in implementation.

6.4 Balaj Saleem

- Non-Functional Requirements, 2 sequence diagrams, and UI mockups in Analysis Report Iteration 1
- Design goals, purpose of the system, boundary conditions, access-control and security, hardware/software mapping in Design Report Iteration 1
- Implementation Process in Final Report Iteration 1
- Functional, Non-Functional Requirements, and 1 Sequence Diagram in Analysis Report Iteration 2.
- Boundary conditions except Failure, access-control and security, hardware/software mapping in Design Report Iteration 2
- Enemy, Bugeye, Crescent, Saucer, SaucerBlades, Spikey, and Slicer classes in implementation.

6.5 Sayed Abdullah Qutb

- Use Case Diagram and the explanation of the use-cases in Analysis Report Iteration 1
- Class Diagram explanations in Design Report Iteration 1.
- Work Done and scenes of the game in Final Report Iteration 2.
- Use Case Diagram, the explanation of the use-cases, and 1 Sequence Diagram in Analysis Report Iteration 2
- Class Diagram explanations in Design Report Iteration 2.
- Pickup, HealthPickup, FuelPickup, Human, Warpzone, FiringBehavior, FireOrangeGlow, FireRedBolt, Button, Explosion, Item, and Point classes in implementation.

7. Glossary

BugEye: These enemies are responsible for picking up humans. After abduction, they move to the top to mutate into Crescent.

Bullet: The spaceship will fire blue bullets which will damage the enemies. It has no cooldown when using it.

Crescent: This enemy results after the mutation of weaker enemies and will only attack the player.

Fuel pickup: The player will be able to utilize a fuel pickup which will restore the space ship’s fuel.

Fuel Upgrades: The player will be able to increase the fuel capacity by buying fuel tanks from the shop.

Health pickup: The player will be able to utilize a health pickup which will restore the space ship’s health.

Health Upgrades: The player will be able to increase the health capacity by buying health tanks from the shop.

Human: The humans on the ground will be abducted by enemies so that the enemy can mutate.

Missiles: The missile is a powerful weapon the player can use that targets the closest enemy and deals more damage than a bullet, it has a cooldown of 5 seconds.

Nuclear Bomb: This weapon will kill all enemies on screen for the player, it has a cooldown of 15 seconds.

Saucer Blades: These are stronger versions of Saucers in terms of speed.

Saucer: This is a low-level enemy that shoots bullets at the player.

Shop: The game will include a shop feature where the player can purchase upgrades.

Slicer: This is a low-level enemy that only attacks the player.

Spikey: These enemies will try to attack the player by getting close to the spaceship and exploding.

Warpzone: Random warp zones generated on the map that will transport the spaceship(player) from one location to another.

References

[1] "usman-kakakhel/CS319-2D-DE: Remake of the Arcade game Defender for CS 319", *Github*, 2019. [Online]. Available: https://github.com/usman-kakakhel/CS319-2D-DE. [Accessed: 22- Dec- 2019]

[2] "CS319 - Group 2D - Project Design Report", *Google Docs*, 2019. [Online]. Available: https://docs.google.com/document/d/1Z0CqleRtj6VfdDHW8i2vrLQkUTko9vSRyHzvspnowQM/edit?ts=5dc1835f#. [Accessed: 22- Dec- 2019]

[3] "CS319 - Group 2D - Project Analysis Report," Google Docs. [Online]. Available: https://docs.google.com/document/d/1ho6j-uEJV7cxFchB4kYFe-pw97rxO_Uo1KD4Rhj6yz0/edit?ts=5dab252c#heading=h.waz2zrxivm6e. [Accessed: 03-Dec-2019].