
University of Dhaka

Department of Computer Science and Engineering

Project Report:
Fundamentals of Programming lab(CSE-1211)

Project Name:
# The Spy

Team Members:
Tasfia Tabassum
Roll:24
Saima Akter
Roll:30

## 1. Introduction

The main game, ‘The Spy’ is a 2D C++ based game made by SDL2 library. It is an offline, single player game.It is a shooting game.The player is playing the spy role. Player have to reach ‘the informer’ to go to the next level.To reach ‘the informer’, player will face some enemies.If the player is able to kill the enemies and able to collect the coin, score will be added.If the player can complete total levels, player will win the game. After the exciting experience of playing the shooting game, to keep the gameplay experience of the players refresh, a simple but interesting mini game was developed to add with the main game. It is called ‘Chibi Spy’ . It is a maze game where the player will operate the chibi character to obtain ‘the pendrive’ and avoid obstacles like fireballs emitting from traps, enemy minions etc. To end the level, the player needs to insert the pendrive into the ‘computer’ by reaching to that point simply.

## 2. Objectives

The objective of the project was to successfully implement the basic SDL and C/C++ knowledge we acquired. We aimed to include many attractive features to make the gaming experience fun. We divided the whole codes into many modules and used header files and structures to make the code easy for understanding and easy for editing in later time. To keep the scores of the players , we used text files to store players’ names and their respective scores. In the case of the maze game, the mazes keep shuffling into random patterns in each level to avoid boredom of the players by repeating the same patterns consecutively. We hope players will have fun by playing this game.

## 3. Project Features

- **The Main Game - ‘THE SPY’ :**

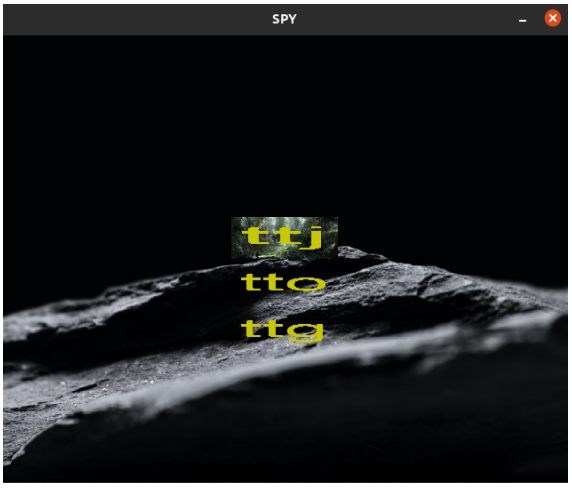
The game has many attractive features.Proper images,characters,and backgrounds were used for a better gaming environment.
Screenshots of some of the features:

**Menu part:**

This is the menu part interface. In menu part there are five options such as “CONTINUE”,”NEW GAME”,”INSTRUCTION”,”HIGHEST SCORES”,”QUIT”.In continue option, if any game was paused previously, the player can resume that game with the help of this part. At most three games can be saved or paused. If the player tries to save/pause more than three games, the latest three saved/paused games will be stored.

## New Game:

When the player clicks on ‘NEW GAME’ option, firstly an interface will open showing the updated list of the latest three players’ names.After clicking on any of them, another interface will open asking the player to enter the payer’s name. The mentioned interfaces are:





In the game, there are a total of three characters.

**The Player:** Players play the role of a spy on a mission to receive information from the informer. To do so, the player needs to navigate in the enemy’s territory and try their best to shoot the enemy characters or avoid colliding with them. At the start of the game, the player has 3 lives. The player can lose life when it collides with the enemy character or if it falls from the route. The character has the ability to move forward, move backward, jump, and shoot.

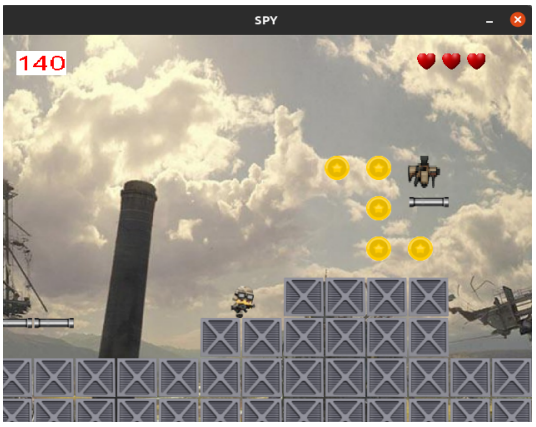
**The informer:** It is a static character. The players can not control this character or it is not run by computer code. This character is stationed at the end of each level indicating to the player of the closure of that level. When the player’s character collides with ‘the informer’ then the player is able to advance to the next level.

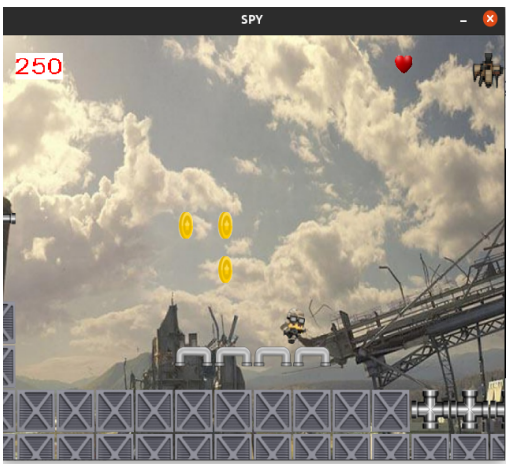
**The enemy:** This character is controlled by the computer code. It's patrolling range is fixed and the velocity for every enemy character is the same. There are multiple enemy characters in one level. The player can shoot the enemy character to make it vanish or can avoid colliding with it to advance forward. But if the player's character collides with an enemy character, then the player will lose one life.

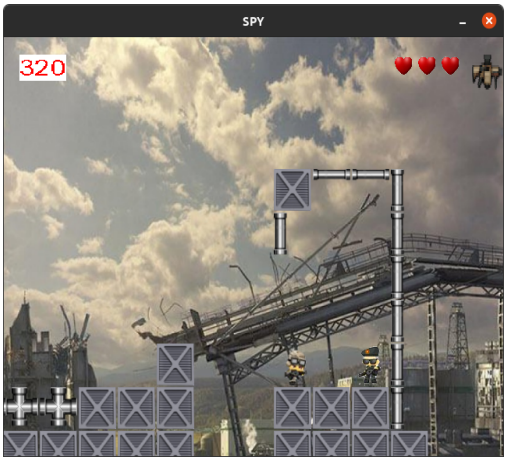
At present, a total of four levels are included in the 'NEW GAME' option. Screenshots of the levels are given below:

★ **Level 1:**

Level1 interface is:

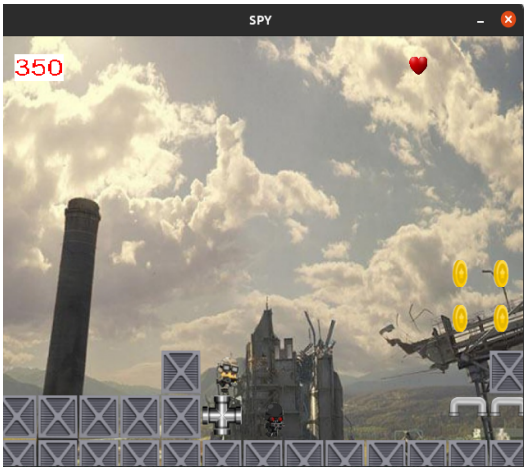



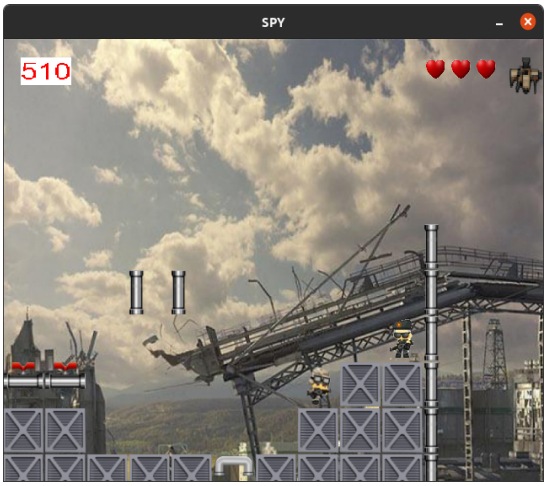



★ **Level 2 :**

Level2 interface is:

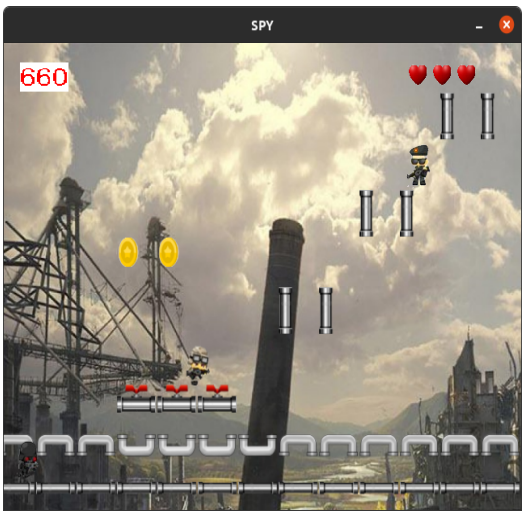



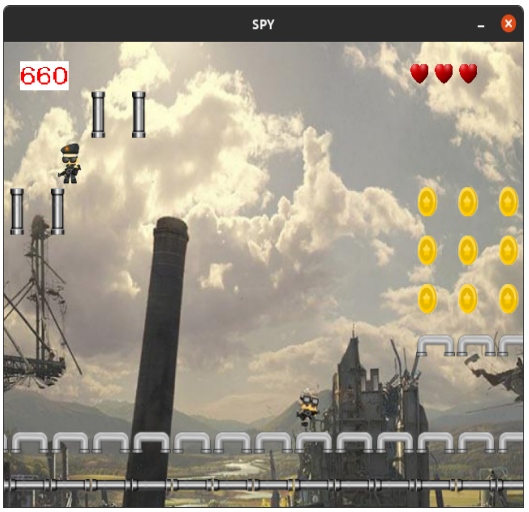


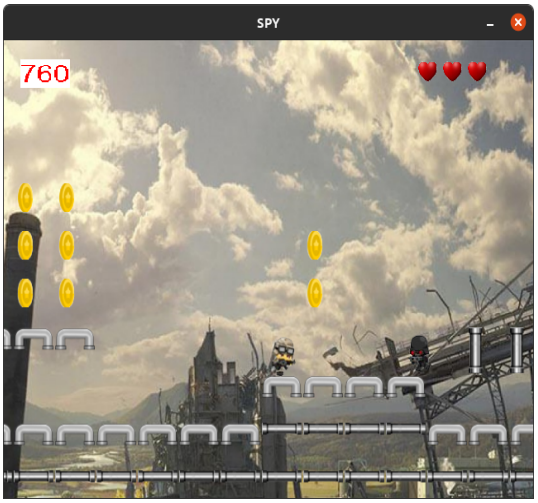



★ **Level 3:**

Level3 interface is:



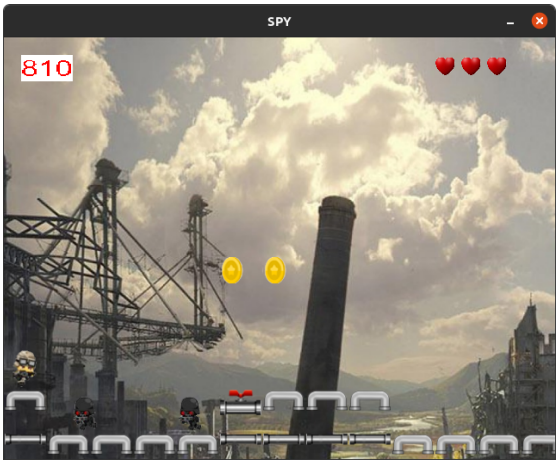



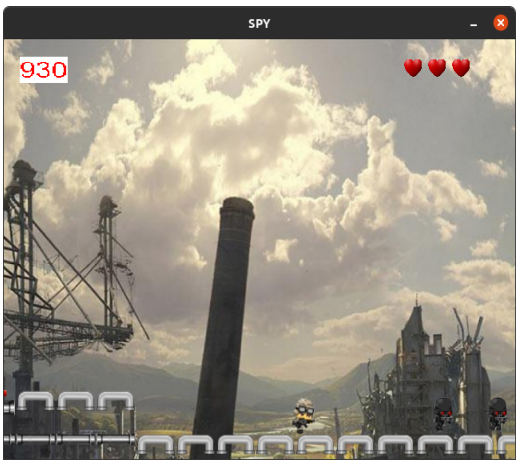



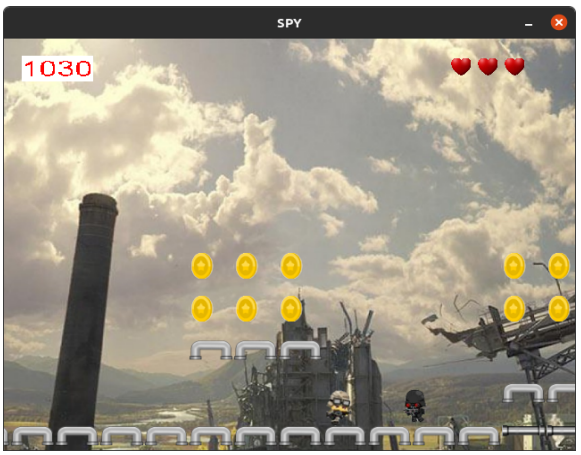


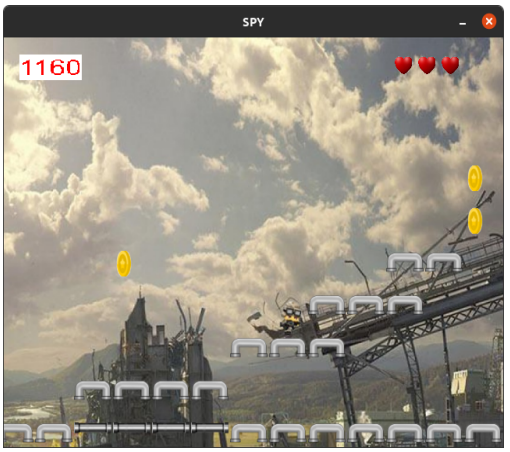
★ **Level 4:**

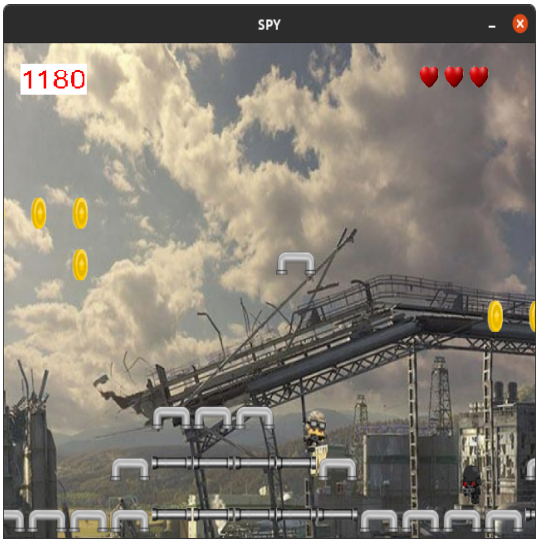
Level4 interface is:

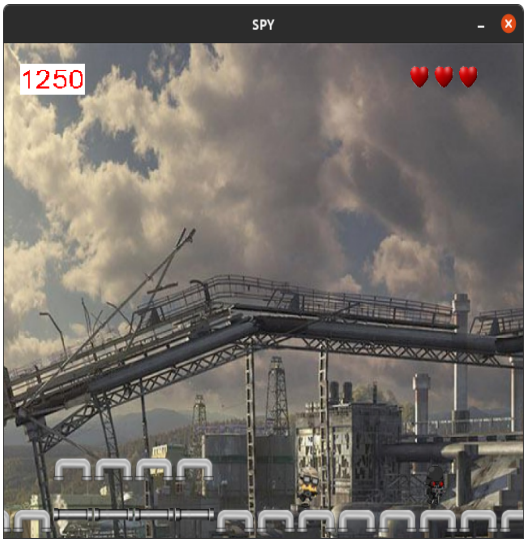


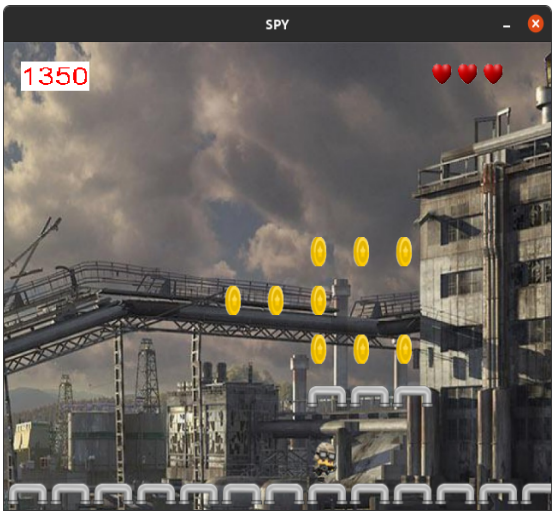



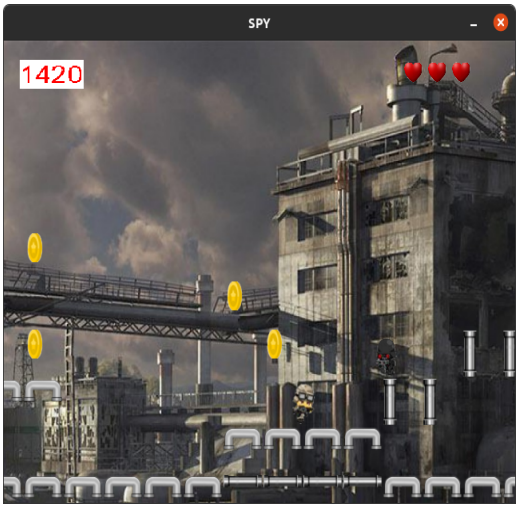

SPY
1030

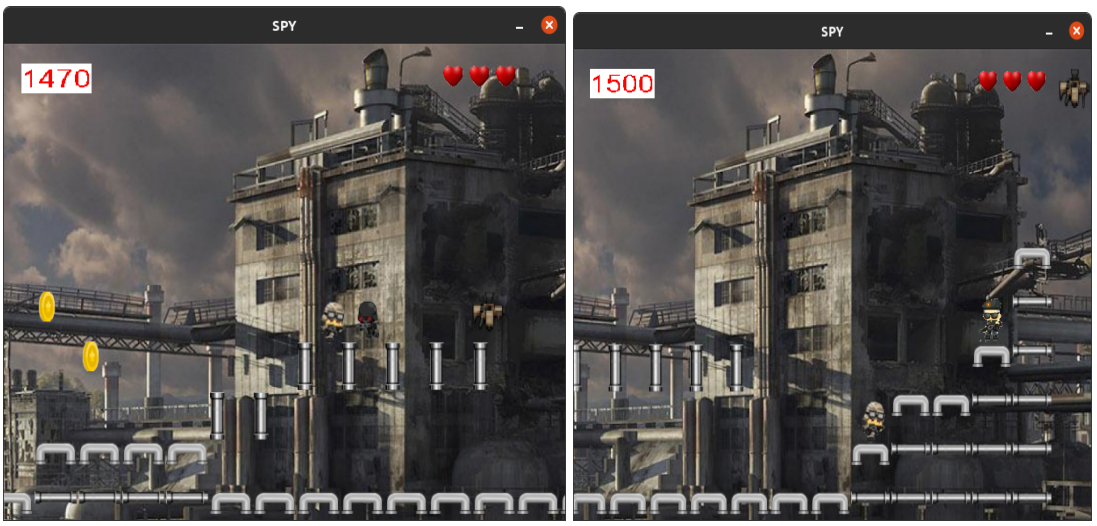

SPY
1160


SPY
1180


SPY
1250


SPY
1350


SPY
1420

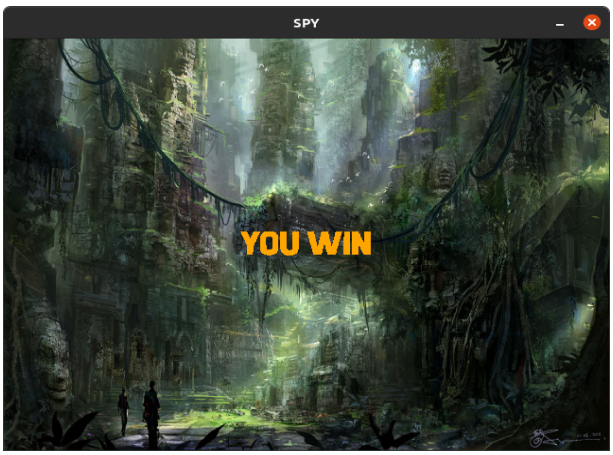


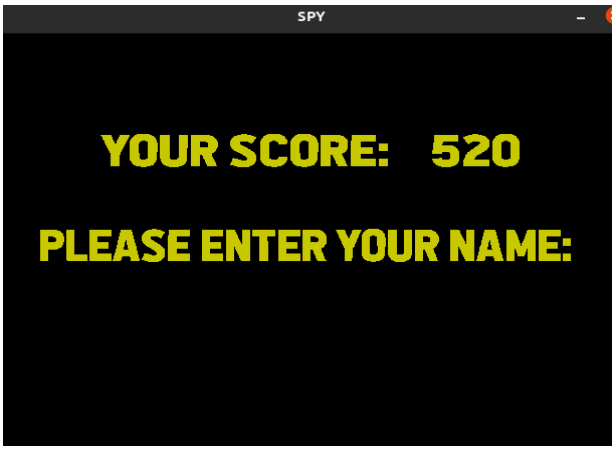

## Game Paused:

Game paused interface is:


## Win Game Interface:

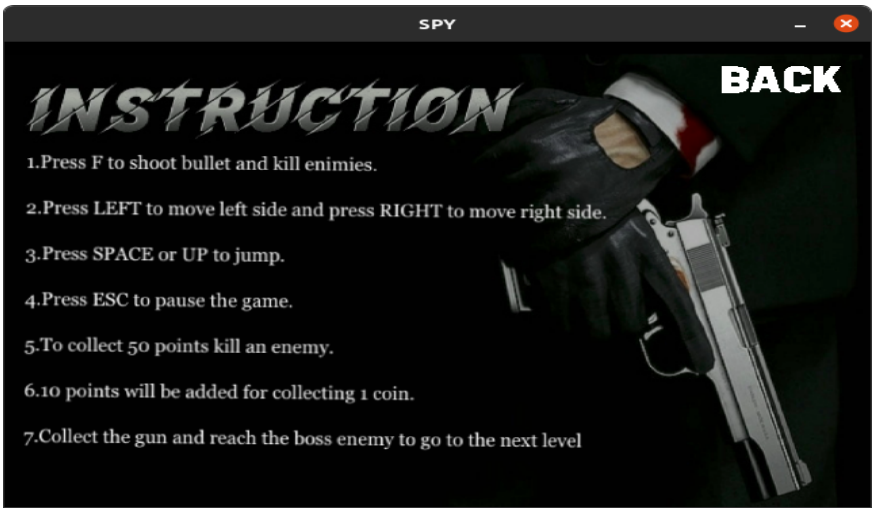
If the player is able to clear all the levels, then the ‘YOU WIN’ interface will show. After that another interface will open showing the obtained score of the player and will ask for the name of the player. After entering the name, this interface will go away and the menu interface will open again.




## Instruction Interface:

Instruction interface is shown below:




To go back to the menu interface, the player needs to press ‘esc’ key of the keyboard.

## Highest scores:

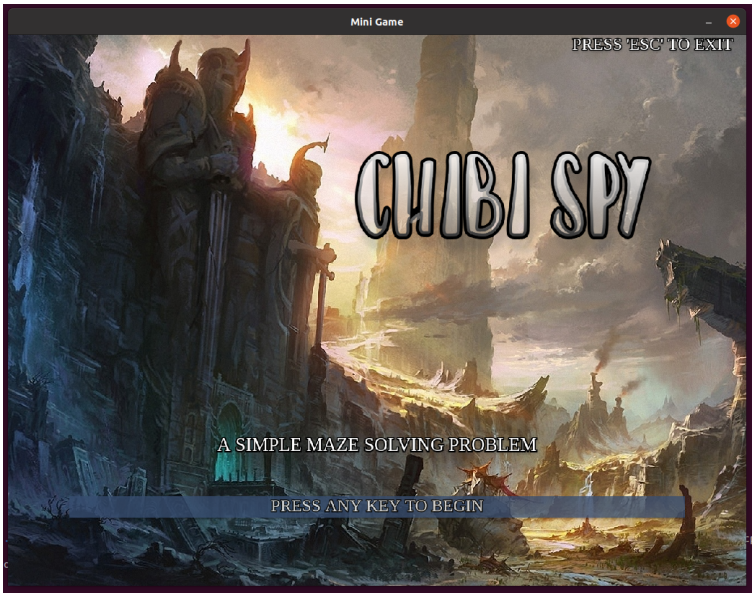
Highest scores interface is shown below:


Only top 5 scores obtained by different players(based on the name provided in the game) will be stored in the ‘HIGH SCORE’ part of the game. To reset the leaderboard, click on the ‘RESET’ option.

- **Mini Game - ‘CHIBI SPY’ :**

Although the game is simple in the way it functions, the game is interesting enough to keep the player entertained. The game includes many interesting features also. Details with screenshots are included below:

## Menu Part:



At the start of the game, a menu interface opens up. If the player press any key of the keyboard except ‘esc’ the new game part will initialize; otherwise if the player presses ‘esc’ key, then the game will end.

## New Game:

In this mini game, there are three character images that can be seen.

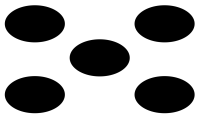
**The chibi spy:** This character is operated by the player. The player can operate the character to move right, left, up and down. The movement is controlled by the arrow keys. The aim of the player is to obtain the ‘pendrive’ and move avoiding the active traps and enemy minions. At the very beginning of the game (level 1), the player starts with 10 lives. When a player loses all of the lives, the game ends.

**Pendrive:** This is not operable by the players. This pen drive in the game contains important information regarding the enemy and the player is to collect it and decode it into the computer. The position of the pen drive changes in each level and throughout that current level, the position doesn’t change. But if a player resets the level, then the position of the pen drive in the level will change.

**Computer:** This character is also not operable by the players. The position of the computer changes in each level and throughout that current level, the position doesn’t change. But if a player resets the level, then the position of the computer in the level will change.

**Enemy minions:** This character is not operable by the players. It’s path is fixed in three points but the minion’s position alternates among these three points following a pattern. If the player’s character collides with the minion directly the the player will lose 1 life and reswans from the starting point of that level.

**Traps:** The game is designed in such a way that the player navigates progressively larger procedurally generated mazes, finding the pen drive to get to the next level while avoiding obstacles such as active traps. There are patterns in which the traps are activated or inactivated. Players can safely pass inactive traps but if the player try pass the active traps, then the player will lose life.

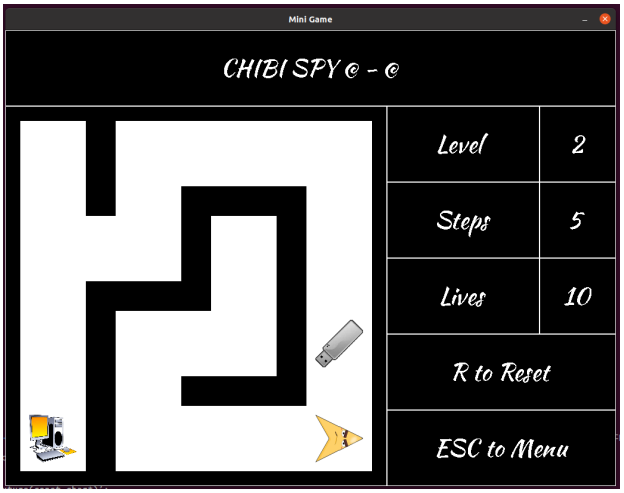


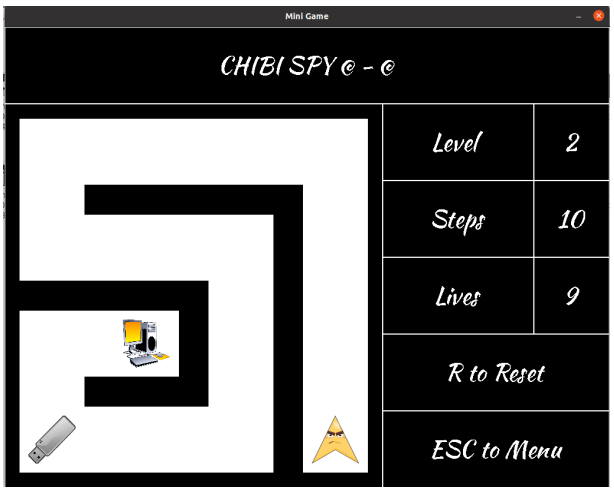
Inactive traps

Active traps

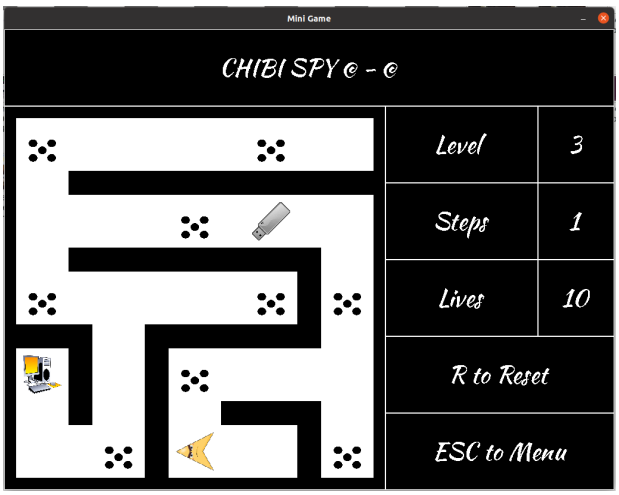
**Maze:** The maze patterns also change in each level or if the player resets a level. The maze gets larger progressively and becomes more complex with more traps and enemy minions. The pattern of the maze is kept random. There are a total of 7 levels in our current game.

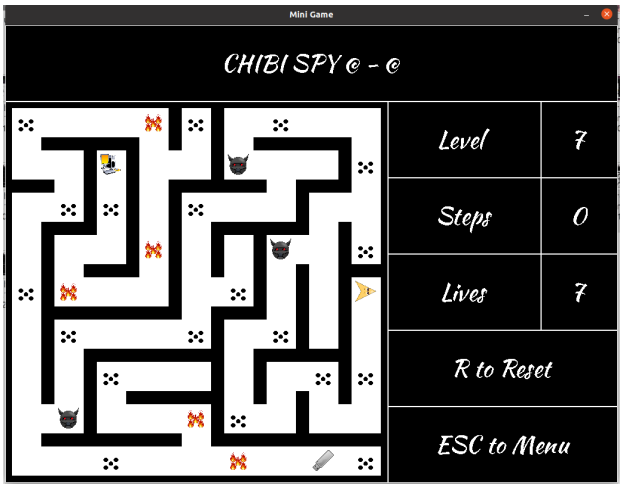
When player resets a level /  even if a player plays the games consecutively the chances of the mazes pattern being same is very slim:



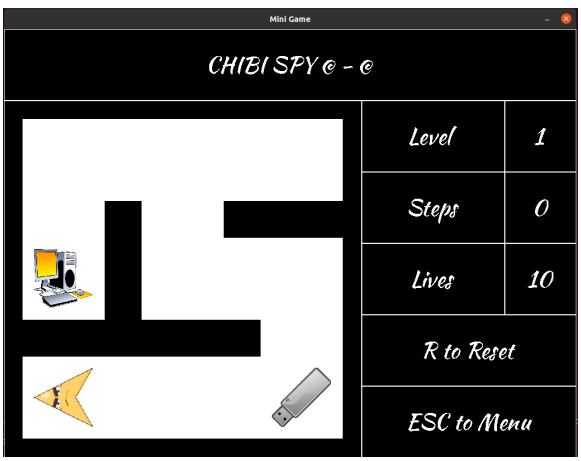

Complexity of the maze increases progressively level by level:

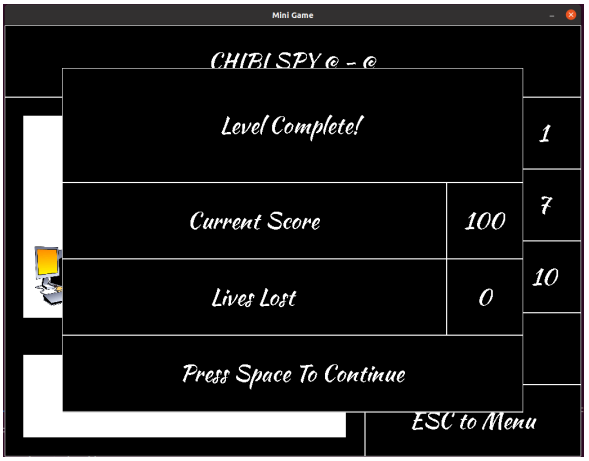


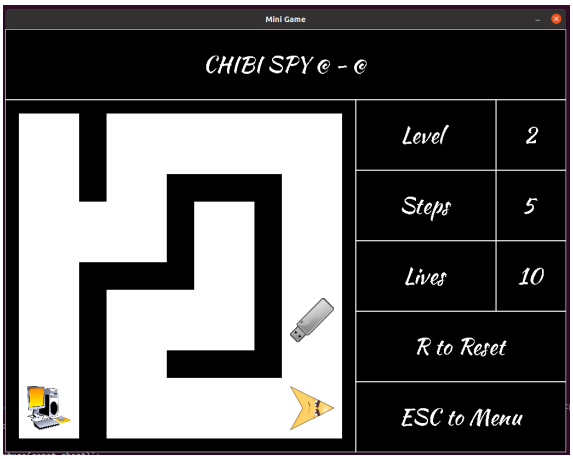

To understand the progressive difficulty of the game, interfaces of the total 7 levels are given below:

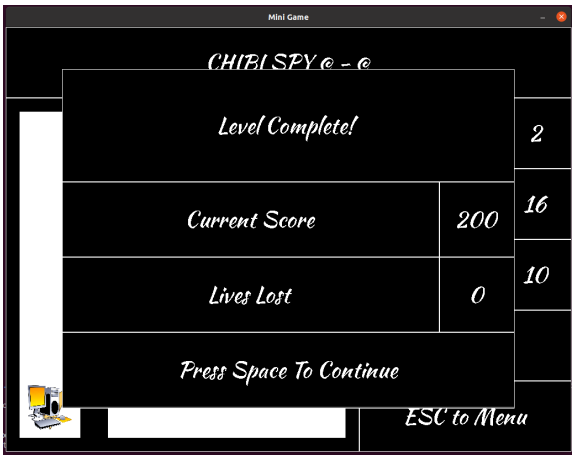
★ **Level 1:**



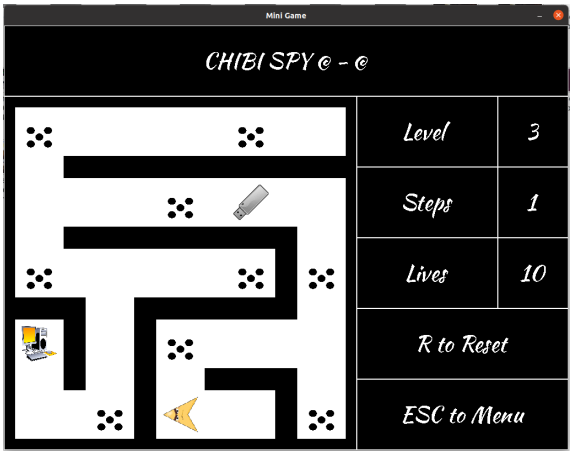



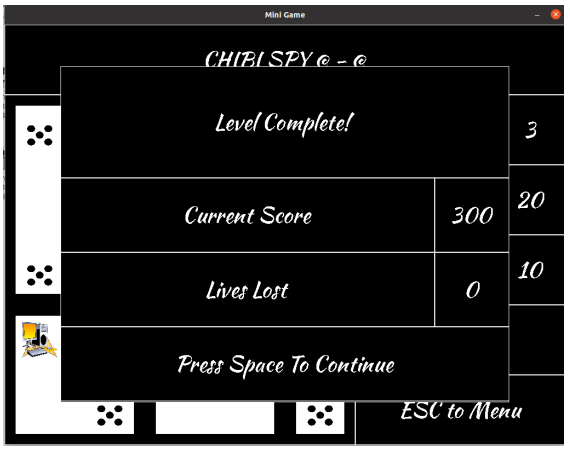

★ **Level 2:**



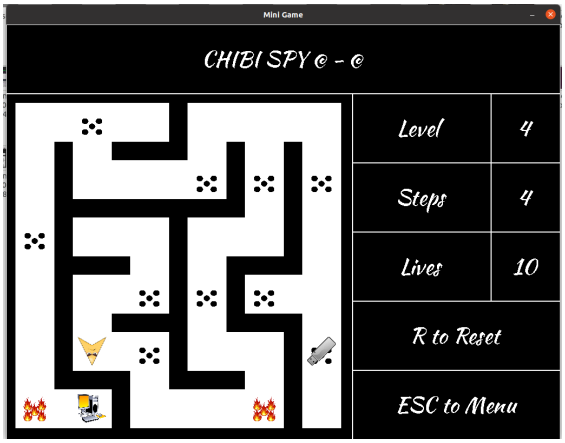



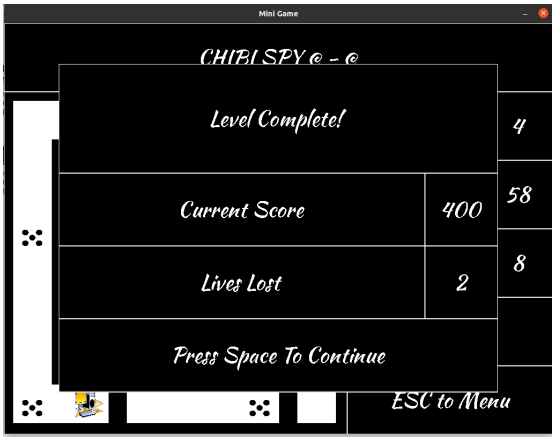
★ **Level 3:**



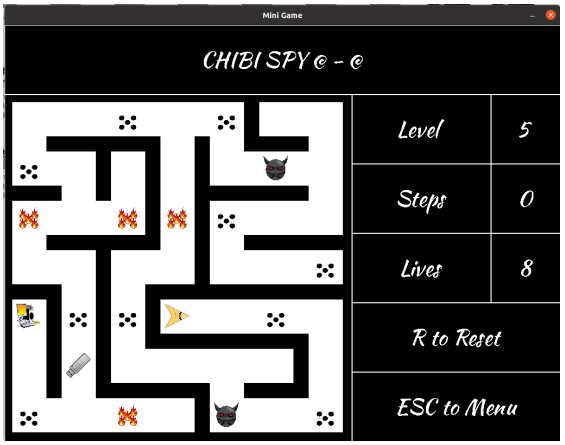



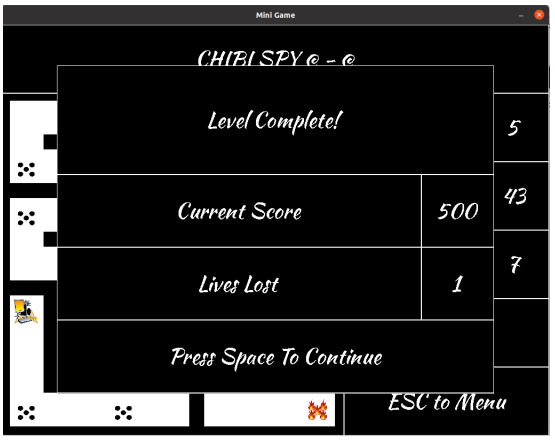

★ **Level 4:**



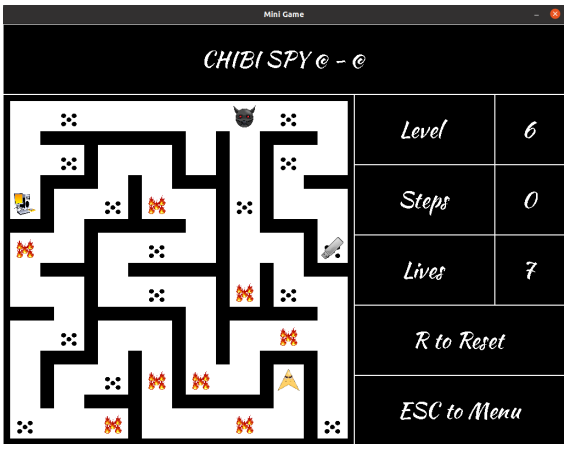



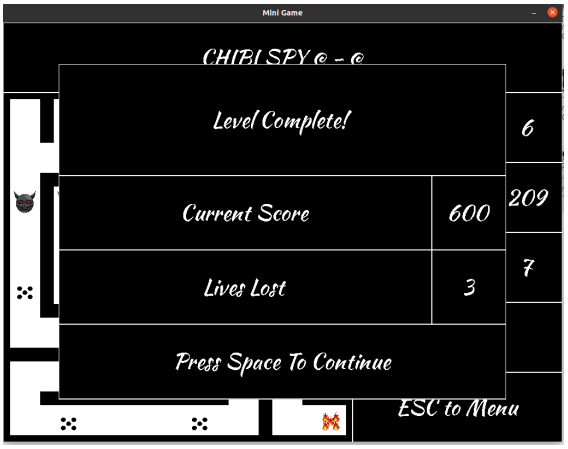

★ **Level 5:**



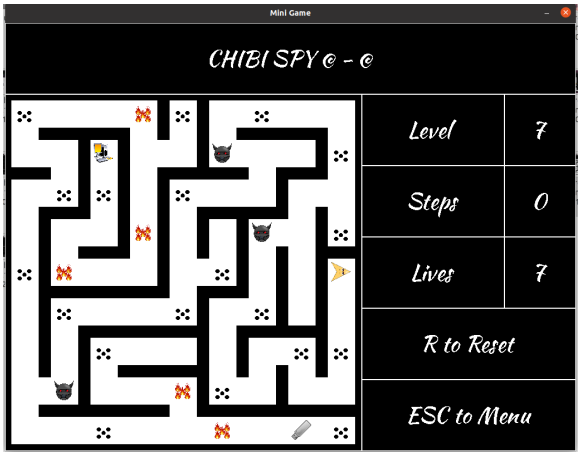




★ **Level 6:**



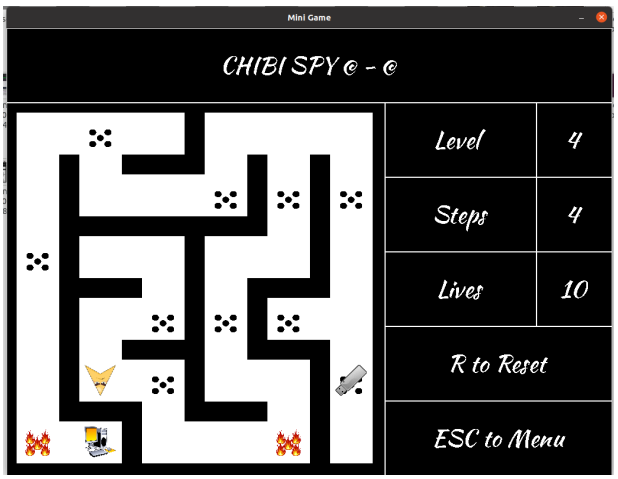



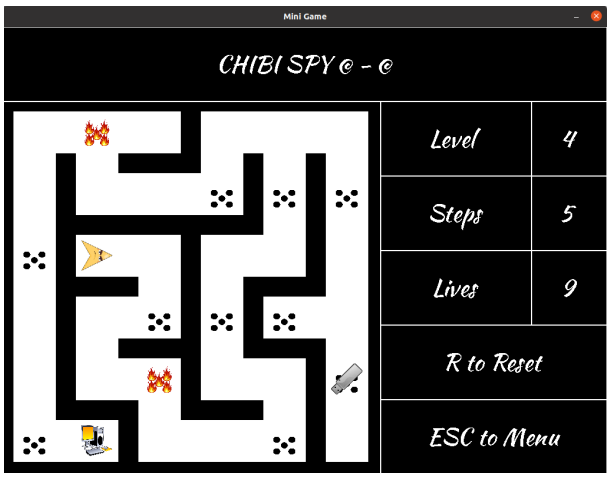

★ **Level 7:**



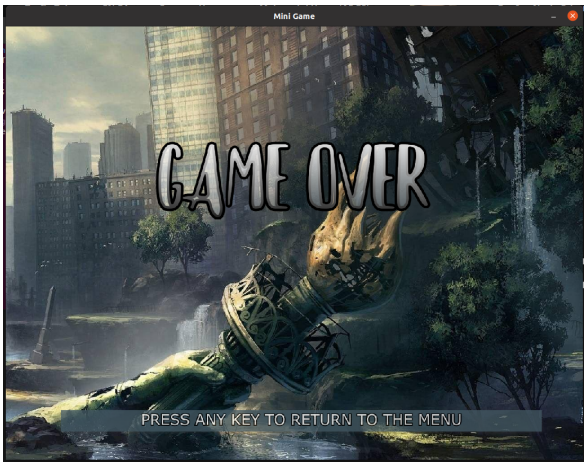

After completing level 7, the game over interface will show.

**Reset:** When the player presses the ‘r’ key of the keyboard in the middle of the game, then the whole level starts anew and the maze reshuffles into a new pattern. The positions of the pendrive, computer, traps, minions change also. The player loses 1 life at the same time.



**Game Over Interface:**




When the player completes all the 7 levels or in case the player couldn't reach the last level but the player has already lost all 10 lives, the game over interface will show up. By pressing any key of the keyboard, the player can return to the menu of the mini game.

## 4. Project Modules

**The Main Game - 'THE SPY' :**

**Player.cpp + player.h -** In the "Player.h" header SDL2/SDL library has been used. "Bullet.h", "collision.h", "physics.h", "score.h" header are used in the "payer.h" header. Class, a user-defined type or data structure is also used in "player.h" header which has some functions for players show,movement, directions,firing,health, shooting bullets,hasGun,handling e.t.c. and access specifiers are public. In the **Player.cpp** file "player.h", "collision.h" header has been called.In this file the functions which have been called in the "player.h" header are implemented.vector also used in this file. Players image size,Players movement,Players direction,controlling players are contained in this module.

**Enemy.cpp + enemy.h -** In the "enemy.h" header SDL2/SDL library has been used. "collision.h", header are used in the "enemy.h" header. Class, a user-defined type or data structure is also used in the "enemy.h" header which has some functions for enemies show,movement,directions,handling e.t.c. and access specifiers are public. In the **enemy.cpp** file the "enemy.h"header has been called. In this file the functions which have been called in the "enemy.h" header are implemented.vector also used in this file. Enemies image size,enemies movement,enemies direction,controlling enemies are contained in this module.

**Bullet.cpp + bullet.h -** In the "bullet.h" header SDL2/SDL library has been used.Class, a user-defined type or data structure is also used in the "bullet.h" header which has some

functions for bullets  show,movements e.t.c. and access specifiers are public.In the **enemy.cpp** file ”bullet.h” header has been called and implemented the functions.

**Collision.cpp + collision.h -** In the “bullet.h” header SDL2/SDL library has been used.Class, a user-defined type or data structure is also used  in the “collision.h” header  which has some functions for collisions and access specifiers are public and protected.In the **collision.cpp** file ”collision.h” header has been called.This module used for collisions..

**Score.cpp + score.h -**In the “bullet.h” header SDL2/SDL and SDL2/SDL_ttf  library has been used.Class, a user-defined type or data structure is also used  in the “score.h” header which has some functions for drawing score ,getting score and access specifiers are public. In the **score.cpp** file “score.h” header has been used and also string used.This module used for score drawing and getting.

**Physics.cpp + physics.h -** In the “physics.h” header SDL2/SDL library has been used.Class, a user-defined type or data structure is also used  in the “physics.h” header  which has some functions for increasing velocity,increasing acceleration,position moving and access specifiers are public.In the **physics.cpp** file “physics.h” header has been used.This module used for increasing velocity,increasing acceleration,position moving.

**Collect.cpp + collect.h -**In the “collect.h” header SDL2/SDL library has been used.”collision.h” header also has been called in the “collect.h” header.Class, a user-defined type or data structure is also used  in the “collect.h” header  which has some functions for collecting frame and access specifiers are public.In the **collect.cpp** file “collect.h” header has been used.This module used for collecting frames.

**Game.cpp + game.h -**In the “game.h” header SDL2/SDL and SDL2/SDL_image library has been used.”collision.h” , “player.h”, “enemy.h”, “collect.h”, “bullet.h”. “Menu.h” header also has been called in the “game.h” header.Class, a user-defined type or data structure is also used  in the “game.h” header  which has some functions for starting game,map loading and  saving, map showing,event handling ,input name,bullet viewing,displacement players,checking frame rate,level loading e.t.c and access specifiers are public.Window,renderer ,some texture and Rect also called in “Game.h” header. In the **game.cpp** file “game.h” and SDL2/SDL_ttf library are also used.string,vector has been used in this module.This module used for controlling total gaming part.

**Menu.cpp + menu.h -**In the “menu.h” header SDL2/SDL,SDL2/SDL_image and SDL2/SDL_ttf library has been used..Class, a user-defined type or data structure is also used in the “menu.h” header  which has some functions for starting new game, starting menu,win screen,resuming menu,highest scores,instructions,score loading,handling arrow,loading media e.t.c and access specifiers are public.window create,some texture and Rect also called in “menu.h” header. In the **menu.cpp** file “game.h”,”menu.h” are used.String,vector,algorithm has been used in this module.This module used for controlling total menu part.

**Main.cpp -** “menu.h” header has been called in this file and menu started.

## The Mini Game - ‘CHIBI SPY’ :

**main2.cpp -** This file is used to initialize the game and it has the gameplay loops. This part handles creating the window and the renderer, rendering the objects and displaying information regarding the game. The ‘int main function’ is the most important part. If ‘initialization’ is successful, then the game goes into the main game ‘while’ loop. This loop is the one which is used to carry out the actual gameplay. Inside this loop, there are four sub-while loops representing different states of the game: Front End, Gameplay, Level Complete, and Game Over. If at any of these points, the player closes the window, the game quits all loops and will end.

The front end loop is very short. Only active when the ‘curGameState == Menu’, it uses the FrontScreen function to display the menu to the player.

The gameplay loop is the largest part. First, the game will display information to the screen (level number, steps, lives, etc.) using the ‘DrawInGameUI function’ and the various ‘DrawText… functions’. After this, it shows the creation of the maze, then waits for player input to determine what to do. Different certain keys are designated to initialize different actions. The player inputs into the game using these certain keys in certain cases, in other times, any mouse press, key press or input will do nothing. Regardless of the player’s choice of input, at the end of the gameplay loop, everything is included to the ‘renderer’ and displayed on the screen using ‘RenderAllGameObjects’. This calls on the ‘Maze’ to add all its object textures (Rooms, Traps, Guards, Keys, Doors) to the renderer and on the Player add its ‘texture’ to the ‘renderer’. It calls the ‘DrawInGameUI’ function to update the information on screen also. When the player obtains ‘the pen drive’ and reaches ‘the computer’ then ‘curGameState’ is then set to ‘LevelComplete’, and we move to the’ Level Complete loop’, which displays the information about how many lives the player lost in that level and the players current score. The score system is that , when the player passes a level, 100 points are added to the previous score points. The player can then press the space key to ‘continue’. This generates the ‘next level’ and returns ‘curGameState’ to ‘InGame’.

If the player runs out of lives at any point, ‘curGameState’ is set to ‘GameOver’, and the game goes into the game over loop. At this point, the player has two options. One is to return to the ‘frontend’ and reset the maze to the first size behind scenes. So if the player then chooses to play again, the game starts anew. The other option is to close the window or press the ‘esc’ key.

**Coordinate.cpp + Coordinate.h -** The position of anything in the maze, including Rooms, any Maze Objects (Traps, Guards, Keys and Doors), and the Player can be known using the coordinate struct.

**Room.cpp + Room.h -** The ‘Room’ class gives us a point in the grid required to build the play area. Each Room has information about its ‘position (roomPos)’, ‘direction of walls (wallDirBit)’, rendering area (roomRect), and texture (curRoomTexture). These help to indicate

where it will be rendered and what it will look like. The maze has a bool ‘inMaze’ that is set active once the Room has been connected to the maze by the ‘Maze script’.

Simultaneously, the room has pointers to other rooms including the adjoining ones (adjRooms), available to connect (availRooms), and already connected (connectRooms). The ‘Maze’ code successfully controls them to form proper connections. Shortly after the ‘Maze’ creates the ‘Rooms’, their ‘adjRooms’ and ‘availRooms’ variables filled with all the rooms the current one is next to. The ‘connectRooms’ variable is filled in later.

‘The wallDirBit’ is used to determine both what sides the room has walls as well as load the proper textures. The rooms that are connected should not have walls between them. So it has a function called ‘CheckAdjRoomDir’ that allows it to determine what direction any adjacent room may be. This allows it to remove that direction from its list of walls.

The Room also has an enum called ‘RoomType’ which has enumerations {None, Start, Final, Key, Trap, Guard }, referring to the different kinds of rooms that could exist. Each room has a vector of these called ‘roomTypes’. When a player enters a room, it will check these types to determine what action must be done (i.e. picking up a key, losing a life, etc.).

**Maze.cpp + Maze.h -** The Maze class creates the actual maze of the game. It begins this process by using the ‘CreateRooms’ function to create a grid of rooms, giving each a unique position, and storing them in a vector called ‘allRooms’. These rooms are initialized as described above, and are not yet in the maze.

The ‘CarveMaze’ function actually goes about creating the maze. It stores one of the rooms at random in ‘curRoomPtr’, puts it into a ‘currentPath vector’, and uses a backtracking algorithm to carve out the maze. It selects one Room at random as the start point (adding 'Start' to that room's roomTypes) and then selects one of its ‘availRooms’ as the next room in the maze, removing the selection from ‘availRooms’ in the process, as well as removing itself from the newly connected room's ‘availRooms’.

It continues this until it encounters a room with no ‘availRooms’, at which point it will pop off the top element of the vector, and check if the previous room in the path had any other available rooms. It continues this until all the rooms are ‘inMaze’.

As the rooms are placed, if the game has gone to a high enough level, the rooms where obstacles spawn will be selected and added to the ‘obstacleRooms’ vector. These are spaced along the path a distance apart set by the ‘obstacleSpacing’. This number needs to be high enough so that a maze with obstacles is traversable. They are later created by the ‘CreateObject’ function, which assigns these to the ‘objectsInMaze’ vector. The type of obstacle created (MazeTrap or MazeGuard) is determined by the current level of the game and the number of connected rooms. Whatever type is created is added to the room's 'roomTypes' variable.

The Maze also keeps track of the longest path. It uses this to find set ‘finalPos’ as the furthest distance away, which is used to set the ‘finalRoom’. A ‘MazeDoor’ object is created there as the way to the next level. After the objects are in place, a ‘MazeKey’ object is created by choosing a

random room that is not equal to the starting or ending position for the maze. As these objects are placed, the associated room's 'roomTypes' variable is changed accordingly.
After the maze is created, it is still used to update the game during every movement. The ‘NextMazeCycle’ function calls the ‘NextCycle’ function in all members of objectsInMaze, causing any objects to perform their behaviors. Additionally, the Maze controls the rendering of all the Rooms and Obstacles within it via its ‘AddMazeRoomsToRenderer’ and ‘AddMazeObstaclesToRenderer’ functions, making each room and object call its own ‘AddRoomToRenderer’ and ‘AddObjToRenderer’ functions, respectively.

**MazeDoor.cpp + MazeDoor.h -** The MazeDoor is the object used to represent the computer in the game that the player has to reach in order to get to the next level. The player needs to obtain the pen drive to activate the computer and reach the next level.

**MazeGuard.cpp + MazeGuard.h -** The MazeGuard is an movable obstacle that the player can run into in the maze. The MazeGuard will only be created next to a room with three connected rooms. The MazeGuard maintains that room as its center ‘(guardCenterRoomPtr)’ while moving between its three connected rooms ‘(`guardCurRoomPtr)’. If the player goes into the room the maze guard is currently at, the player will lose a life and respawn at the starting point of that level if the player has enough lives.

**MazeKey.cpp + MazeKey.h -** The MazeKey is the object used to represent the pen drive in the game which is used to activate the MazeDoor (here the computer) to get to finish the current level.

**MazeObject.cpp + MazeObject.h -** The MazeObject class is inherited by numerous other classes (Player, MazeTrap, MazeGuard, MazeDoor, MazeKey). This is fundamentally used to store the data that would be common amongst any object that appears in the Maze, namely a ‘position (objPos)’ a pointer to its ‘room (curObjRoom)’, its ‘on screen location (objRect)’, its ‘current texture (curObjTexture)’, and a pointer to the ‘renderer (`objRenderer)’. It has ‘SetObjRoom’ and ‘SetObjectRect functions’ for setting the’ object's room’ and ‘rect variables’, as well as a function to ‘AddObjToRenderer’. Lastly, it contains a virtual function ‘NextCycle’ that certain derived classes may use to implement what they do during each cycle of the game.

**MazeTrap.cpp + MazeTrap.h -** The MazeTrap class defines a hurdle that the player can run into in the maze. The trap activates based off of its ‘curTrapTime’ and ‘maxTrapTime’. The ‘curTrapTime’ is incremented by ‘NextCycle’ (every game cycle). After it becomes larger than the ‘maxTrapTime’, it resets to ‘0’ and activates the ‘Trap’, changing its texture for ‘rendering’ and making the room damazing for the player.

**Player2.cpp + Player2.h -** The Player2 class creates the controllable player to the maze. When each level starts, the player has a ‘startRoom’ variable set, so that the player knows where to begin and where to respawn if they die in that level. Additionally, the player ‘bool hasKey’ is set to false.

When the ‘PlayerMove’ function is called, it returns a boolean ‘successfulMove’ based on if the movement is possible or not. If the player moves in the direction of a wall, then the movement will fail, and the player will only change its texture to face the proper direction. If the player moves in the direction of a ‘connectRoom’ relative to the player's current room, the move is successful.

If the move was successful, the main.cpp class has the player call its ‘CheckForObjects’ function, with which the player checks if there is any object in the same position. If there is a key(in game, it’s pen drive), ‘hasKey is set to true. If there is an obstacle (Trap or Guard), its ‘playerLives’ variable is decremented and it will restart at the ‘startRoom’. The player may also lose a life if the ‘R’ key is pressed, causing the maze and player to reset and costing the player one life.

## 5. Team Member Responsibilities:

**Main Game - THE SPY :**

**Player.cpp + player.h -** Saima Akter.
**Enemy.cpp + enemy.h -** Saima Akter.
**Bullet.cpp + bullet.h -** Saima Akter.
**Collision.cpp + collision.h -** Saima Akter.
**Score.cpp + scoe.h -** Saima Akter.
**Physics.cpp + physics.h -** Saima Akter
**Collect.cpp + collect.h -** Saima Akter.
**Game.cpp + game.h -** Saima Akter.
**Menu.cpp + menu.h -** Saima Akter
**Main.cpp -** Saima Akter
**Graphics related works** - Tasfia Tabassum.

**Mini game - CHIBI GAME**

**main2.cpp -**Tasfia Tabassum.
**Coordinate.cpp + Coordinate.h -** Tasfia Tabassum.
**Room.cpp + Room.h -** Tasfia Tabassum.
**Maze.cpp + Maze.h -** Tasfia Tabassum.
**MazeDoor.cpp + MazeDoor.h -**Tasfia Tabassum.
**MazeGuard.cpp + MazeGuard.h -**Tasfia Tabassum.
**MazeKey.cpp + MazeKey.h -** Tasfia Tabassum.
**MazeObject.cpp + MazeObject.h -**Tasfia Tabassum.
**MazeTrap.cpp + MazeTrap.h -**Tasfia Tabassum.
**Player2.cpp + Player2.h -** Tasfia Tabassum.
**Graphics related works** - Tasfia Tabassum.

## 6. Platform, Library & Tools

The game is developed in Linux ‘Ubuntu’ environment and the game source code was written in C++ language. In C++ language, the included libraries are: isostream vector, algorithm, string, fstream etc. SDL2/SDL was used to build up this project. SDL2 , SDL2_image , SDL2_ttf libraries were implemented in this project. Sublime text, Atom , GIMP, Inkscape, Kdenlive, TexturePacker , Pixir were some of the tools used to develop the project and its related works.

## 7. Limitations

Although we were able to learn many new topics and information, there were certainly many more we could have learnt and applied if there was no time constraint. Therefore there are certainly many limitations which we wish to remove and develop the project in the future. Although we have successfully developed a main shooting based game and a mini game to complement the whole project, even after trying our best at our present level, we were unable to successfully incorporate the mini game in the main game menu. In the main game, we further wish to include the ‘respawning’ feature which we currently could not include due to limitations. Also if we get more chances, variations in the map and environment could be added in the future which is somewhat lacking in our current project. These are some of the things we felt that are limitations in our project.

## 8. Conclusions

After completing the project , we now have a clear basic knowledge of the usage of functions, structures, modules and files. To develop the project we also got to know about SDL2 libraries.Before the project, we were completely new to SDl. Now we are much more comfortable with working with SDL and making simple games with it. We plan to continue developing this project to make it more efficient and add many more features to it. Although we expected to finish the project much earlier and efficiently with all the plans working in our favor, experience has taught us many things. As mentioned before, even when we faced setbacks and there were many limitations, team work has made many things possible for this project to develop to this state. The whole experience was a new and interesting one. We hope to learn from this experience and proceed further.

## 9. Future plan

1. Add the feature of ‘respawning’ in the game.
2. Add the feature to ‘on/off’ game music
3. Further develop game levels with different ingame options
4. Gradually increase difficulty level
5. Add introductory video in the game
6. Add mini games to make the game more fun

# *Repositories*

GitHub Repository:https://github.com/Saimatonni/THE_SPY_game_2D_SDL2

Youtube Video: https://youtu.be/Z_ouQrx90xE

# *References*


- http://lazyfoo.net/tutorials/SDL/?fbclid=IwAR2O1ShzC2EcgSVZhWPTE5LHjCD1T0wTji4Vd0F855UHq5RYTNKQ1U2wG8c
- https://www.youtube.com/watch?v=FCRmIoX6PTA
- https://www.parallelrealities.co.uk/tutorials/#shooter
- https://www.libsdl.org/
- http://www.sdltutorials.com/sdl-maps?CommentsPage=0
- https://www.tutorialspoint.com/cplusplus/index.htm
- https://www.youtube.com/watch?v=q06uSmkiqec
- https://www.youtube.com/watch?v=EvAzVhAii_o
- https://www.youtube.com/watch?v=L0NJHQ27qHo