

University of Brighton
Computer Games

CI411 - Introduction to Game Programming 2021 -22

Coursework 2: C++ SDL Game

Module Leader: David Dorrington

Student: Julia Kolano
Date: 05/05/2022

Contents

Contents	1
1. Introduction	2
2 Game Design	3
2.1 Type and Style of Game / Genre	3
2.2 Screen Layout and Format	3
2.3 Control System	4
2.4 Gameplay & Core Rules	5
2.5 Graphical User Interface	5
2.6 Game Designs	6
3. Implementation	7
3.1 Game Objects Diagram	8
3.2 Program Structure	8
3.3 Player Controlled Objects / Character	9
3.4 NPCs	10
3.4 Objects	11
3.5 Interface	13
3.6 Game Management	14
4. Testing, Problems & Solutions	15
5. Critical Review	19
6. Conclusion	20
Bibliography and References	20
Graphical Assets	21
Audio Assets	22
Appendices	23
How to play	23

1. Introduction

The overall purpose of this document is to explain the design process and implementation of producing a simple prototype of a 2D game using the C++ programming language.

In the opening section of this document, the first part of developing a 2D game that will be discussed is game design. In this section, what will be considered is the importance of producing a well thought through design before starting the development stage. Furthermore, the various sub-sections in this part of the report will discuss the process of designing each of the levels of the game in detail, as well as include multiple sketches and diagrams that will help to visualise the thought process behind the design. Additionally to the graphical aspects of the design, this section will also go over the design process behind the control system of the main player controlled character, and the overall gameplay rules.

The next segment of this report will cover the development and implementation of the program that was used to create the final game. This part will include respective sub-sections that will analyse all of the objects and characters that appear throughout the game, and their functionality in the game. Furthermore, the section will also cover the interface and the game management, such as game loops within the main function. There will also be an explanation for how some of the variables control the flow of the game, and how they affect its replay ability.

Finally, in the end section of the main body of this report, testing of the game that was implemented throughout the development of the game will be covered in detail. All the problems that were found throughout the development will be analysed and provided with potential solutions, so that the final implementation of the game could be left in the most professional state possible.

2 Game Design

This section will cover the most important design decisions that have to be considered before making the game. “Design is the process of intentionally creating something while simultaneously considering it’s objective (purpose)” (Schaefer, 2015). This implies that preparation of the design before implementing features into the game massively speeds up the workflow, and decreases the likelihood of problems occurring during development.

Therefore, the game that will be discussed in this report is a game that is similar in genre and in art style to the original ‘Final Fantasy’ and ‘The Legend of Zelda’ games. The name of the game covered in this report is ‘Treasure Quest’, due to the fact that the main objective of the game is to find a hidden treasure, and the main character embarks on a quest to find it.



2.1 Type and Style of Game / Genre

The genre of ‘Treasure Quest’ is a fantasy adventure role playing game. The main feature of an RPG is “a protagonist or hero who goes through different levels to improve their skills and defeat enemies” (Jordan, 2021), moreover the adventure genre “emphasizes exploration through a virtual landscape” (Baldwin, 2017). Hence ‘Treasure Quest’ has the purpose of controlling a main protagonist, killing enemies, and exploring different landscapes, as well as including supernatural characters, such as elves and orcs, it can be considered a fantasy adventure RPG.

Furthermore, the visual style of the game, with reference to ‘3-D Ace’, an art outsourcing studio, is a combination of both vector and pixel art with fantasy elements. This is due to most of the objects and backgrounds for the game being made using vector art, with some of the sprites made using pixel art. Finally, the fantasy elements come from the backgrounds and characters being inspired by folklore and mythology.

2.2 Screen Layout and Format

The overall screen that the game occupies is covered by an area of 800 pixels in width and 600 pixels in height. So the resolution of the screen is 800x600, which is equivalent to a 4:3 aspect ratio.

The playable area covers the whole of the 800x600 pixels, however to accommodate for the GUI that is also present on the screen in the same playable area. Some objects were created within the

game that prevented the PC from accessing certain areas, including the areas where the GUI was displayed.

The view that the user has while playing 'Treasure Quest' is a top-down view, where the player looks at the character from the top at an angle, which allows for the main character to move in all directions. However, some of the levels might give an impression that the PC is viewed from the side, therefore it's up to the interpretation of the player.

Furthermore, each level is constrained to one viewport, so moving off the edge of the screen will move the PC onto the next level. However, only under certain conditions are met, if they're not, then the character is allowed to move from one side of the screen to the other when moving out of the range of the viewport.



Figure 2 - Screen Layout of the game

2.3 Control System

The control system that this game was designed for is a keyboard based system, not including the mouse. This means that everything that the game needs to interact with this game is a keyboard, however the mouse controls has not been implemented for the game. However, this could be a feature that could be added in future development.

Control	Escape button	'W' button	'S' button	'A' button	'D' button	Space button	'1' button	'2' button	'E' button	'P' button
Action	Exits the game	Moves PC up	Moves PC down	Moves PC left	Moves PC right	Fires the arrow	Starts the music	Stops the music	Opens the chest	Restarts the game

2.4 Gameplay & Core Rules

The intended style of gameplay for 'Treasure Quest' is to explore the areas provided by each level, and complete objectives in each of them to be able to advance into the next level, and finally complete the game.

The main goal of the game is to retrieve the treasure hidden in a chest that appears at the end of level 3. This is the only main goal necessary for the completion of the game. However, there are more states that will prevent the player from reaching the goal, and consequently failing the game, having to start again.

One of these states is the amount of lives the player has. There are multiple objects and enemies that will reduce the amount of lives of the PC, so if the lives drop down to zero, then the player fails. Similarly, another state is the amount of air that the PC has at the beginning of level 1, when the PC is underwater, if the air runs out before the player completes the level, the PC will die and the player will have to start the game over again.

Furthermore, each of the levels have a main objective that need to be fulfilled by the player for them to be able to advance to the next section.

In level 1 the main goal is to find a key that is hidden within the seaweed at the bottom of the screen. However, there is also a secondary objective, which is to avoid the fish that are swimming around, since they deal damage to the character upon contact.

In level 2, the main objective is to move the boulder out of the way, so that the entrance to the building can be cleared and the key can be used to open the door and advance to the next section.

Finally, the challenge of level 3 is to kill all of the enemies that are advancing onto the PC. Then the next objective of the player is to open the chest and retrieve the treasure inside it. That's the way to successfully complete the game.

2.5 Graphical User Interface

The GUI is a critical part of any game, because it makes the player aware of all of the states that matter to them, and that they have to control to successfully advance through the game.

Therefore, for 'Treasure Quest', it was decided that the most important states of the game for the player are the amount of lives they have left, these being updated automatically every time the PC takes damage.

Furthermore, the equipment is also shown on the screen, so that the player is aware if they've picked an important item up, or if they have used any of the items during their play through.

Finally, the amount of the air left is also displayed on the screen, but only during level 1. This was done so that the player is aware when they are underwater, and when the character is walking on land. Making it easier for the player to recognise the risk of the PC drowning if the air runs out.



Figure 3 - Graphical User Interface of 'Treasure Quest'

2.6 Game Designs

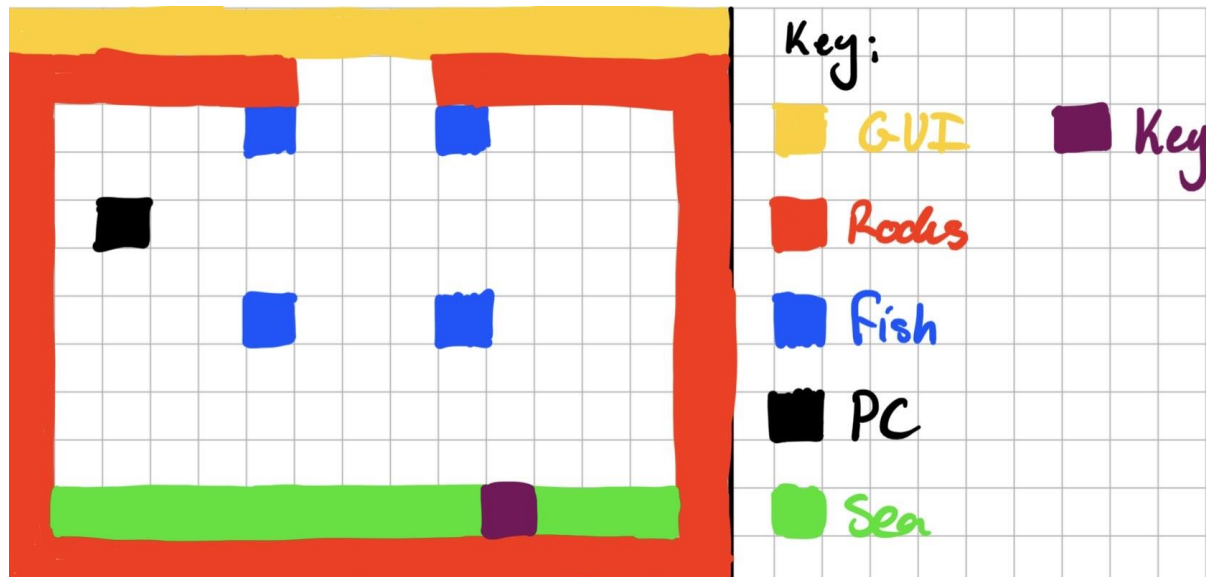


Figure 4 - Level 1 Layout Sketch



Figure 5 - Level 2 Layout Sketch

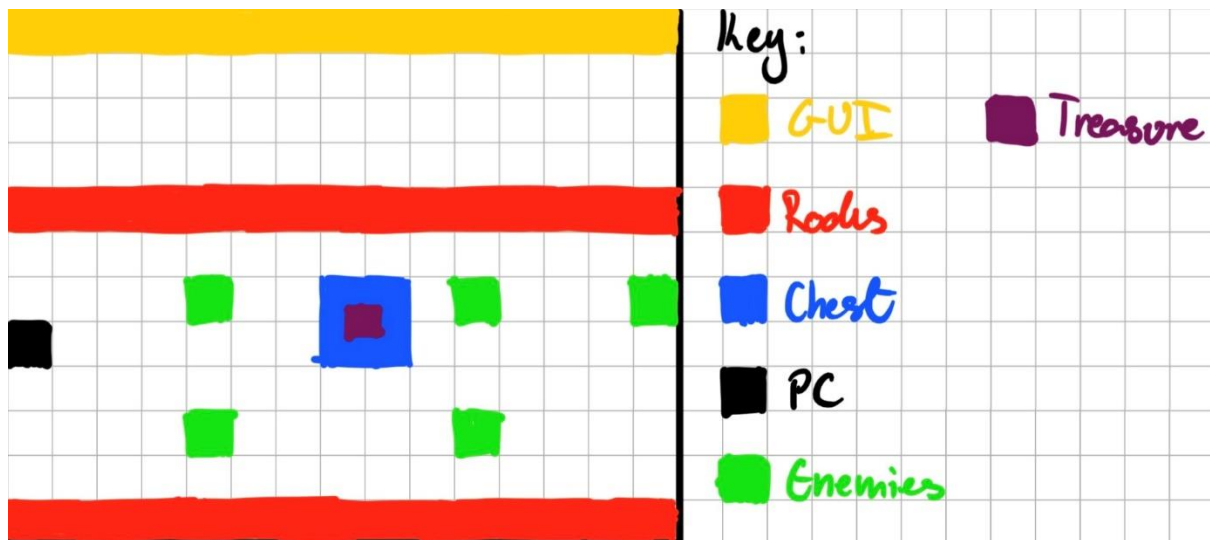


Figure 6 - Level 3 Layout Sketch

3. Implementation

The game titled 'Treasure Quest' is a 2D RPG game that was written in the C++ programming language. Furthermore, the technology that was used alongside with C++, was the SDL2 development library, which is "a cross-platform development library designed to provide low level access to audio, keyboard, mouse, joystick, and graphics hardware" (Simple DirectMedia Layer - Homepage, 2022).

3.1 Game Objects Diagram

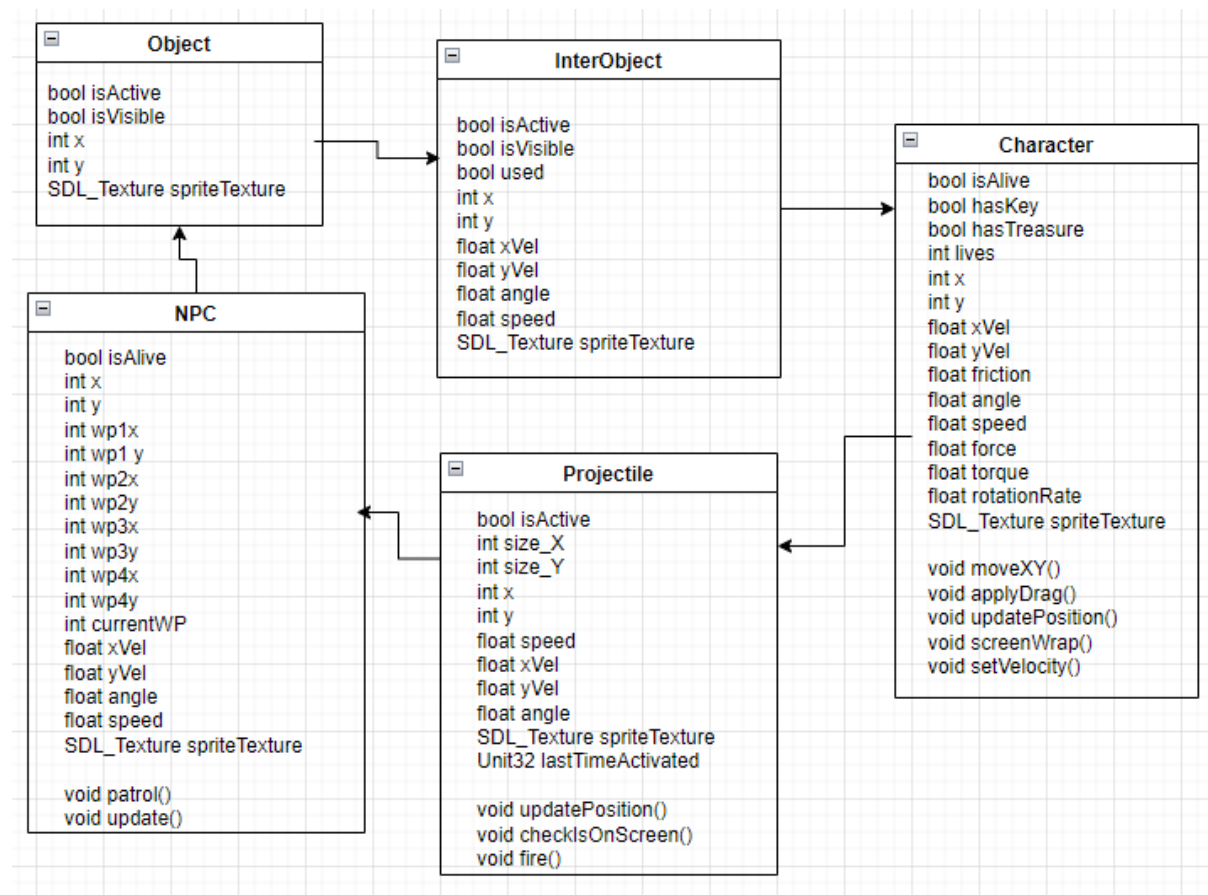


Figure 7 - Object diagram for 'Treasure Quest'

3.2 Program Structure

The program for the game was structured in a way that made it as easy to read as possible. Therefore, the program was split into three separate files: 'GameObjects.cpp', 'GameObjects.h', and 'SDLGame.cpp'. The 'GameObjects.h' header file includes all of the classes, which are then used in the main C++ file, called 'SDLGame.cpp', to make several game objects. Similarly, the 'GameObjects.cpp' C++ file includes all of the functions used by the different classes.

Furthermore, the main C++ file includes all of the initialisations of functions, objects, and some variables that didn't fit with any of the initialised objects. This file also includes other functions that were important for the overall flow and logic of the main function.

The main function determines the overall structure of the game, and conducts its game loop. Firstly, the main function calls another function that initialises all of the parameters needed for a correct implementation of the SDL2 library. Then, the function enters the main game loop by starting with calling a function that generated all of the object sprites and loads the first level of the game. The function then enters a secondary loop, where all the functions that control and monitor the states of the game are called. After, this secondary loop is exited, another function is called that will load the next level if all of the conditions have been met. Then, the loop continues again until the character dies, or if the player successfully reaches the goal. When this happens the core loop is exited and the game ends, with one last function being called that deactivates the SDL2 library.

```

int main(int argc, char* args[])
{
    Uint32 frameStartTime;

    startSDL();

    //Main Game Loop
    while (replay)
    {
        createLevel();

        while (gamePlaying)
        {
            frameStartTime = SDL_GetTicks(); // time since starting in milliseconds

            playerInputHandler();
            updateGameObjects();
            checkGameStates();
            updateScreen();

            limitFrameRate(frameStartTime);
        }

        LevelCompleteScreen();
    }
    closeSDL();
    return 0;
}

```

Figure 8 - Main function and core loop for 'Treasure Quest'

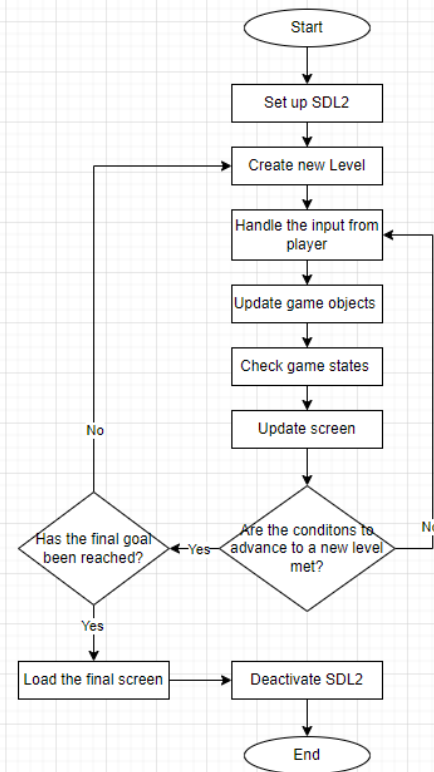


Figure 9 – Flowchart representing the core loop of 'Treasure Quest'

3.3 Player Controlled Objects / Character

The character is being controlled by the use of the player's keyboard, they can use the 'W', 'A', 'S', and 'D' keys to make the character move around the screen. They can also use the space button to make the character fire an arrow that can destroy some objects and kill enemies.

On the other hand, the main variables, that take care of what state the character is in, are 'isAlive' and 'lives' variables. These keep track of how many lives the character has, and is the PC is alive or not. Furthermore, the 'hasTreasure' and 'hasKey' variables keep track of the character's equipment and they help when the character want to interact with the 'Key' and 'Treasure' objects. Finally, there are also some variables that simulate the physical aspects of the character's movement. Such variables are 'xVel' and 'yVel' that control the velocity of the character for both of the axes.

Furthermore, the Character class that the PC belongs to has access to some of functions that help with the actions that the PC can perform. For example, the 'moveXY' function helps to move the character based on the player's keyboard input. Working along with that function is also the 'applyDrag' function, which helps to slow the character's movement slightly, so that it doesn't keep accelerating forever. Lastly, the 'screenWrap' function was also modified specifically for the 'Character' class due to the fact that the sprite for the PC had different measurements in the x axis

to the one's for the y axis, so that a different amount of the sprite disappeared before the character actually moved to the other side of the screen.

```
if (SDL_HasIntersection(&pcRect, &keyRect))
{
    key.isVisible = false;
    pc.hasKey = true;
}
```

Figure 10 - The 'hasKey' variable implementation

```
void Character::screenWrap()
{
    // Screen Wrap to opposite side if sprite leaves screen
    if (x > SCREEN_WIDTH) x = 0;
    if (x < 0) x = SCREEN_WIDTH;
    if (y > SCREEN_HEIGHT) y = 0;
    if (y < 0) y = SCREEN_HEIGHT - 50;
}
```

Figure 11 - The 'ScreenWrap' function for the Character class

3.4 NPCs

The NPCs that were included in 'Treasure Quest' were the enemies in level 3 and the fish in level 1. The NPC class, that both the fish and enemies are under, shares a great number of similarities with the Character class, except the fact that it doesn't include the variables or that were used for controlling the main character, and include some class-specific functions, such as 'patrol' and 'update' functions.

Both of the enemy and fish NPCs utilise the 'update' and 'patrol' functions, but each for their own purposes. As an example, the enemies use these functions for targeting the PC and following it, so that they can attack the PC. However, the fish utilise the same function, however they also make use of some additional variables from the NPC class to move in a pattern around the screen. The fish use the 'patrol' function with passed waypoint objects as parameters, so that they could follow these waypoints, instead of the PC, like the enemy NPCs do.

Furthermore, a supplementary variable was created outside of the NPC class, called 'enemiesLeft'. This variable was created for the purpose of keeping track of how many enemy NPCs are left on screen, so that the conditions for a chest object appearing on screen can be met.

```
pc.lives -= 1;
enemiesLeft -= 1;
enemies[i].isAlive = false;
```

Figure 12 - The 'enemiesLeft' variable implemented upon the NPC's contact with the PC

```
enemies[i].isAlive = false;
bullet.isActive = false;
enemiesLeft -= 1;
```

Figure 13 - The 'enemiesLeft' variable implemented upon the NPC's contact with the arrow object

```
for (int i = 0, j = 0; i < 3; i++, j++)
{
    enemies[i].isAlive = true;
    enemies[i].x = 200 + j * 250;
    enemies[i].y = 250;
}
for (int i = 3, j = 0; i < 6; i++, j++)
{
    enemies[i].isAlive = true;
    enemies[i].x = 200 + j * 300;
    enemies[i].y = 400;
}
```

Figure 14 - The 'isAlive' variable implemented for setting up the initial positions of the enemy NPCs in level 3

```
for (int i = 0; i < sizeof(enemies) / sizeof(enemies[0]); i++)
{
    if (enemies[i].isAlive) // exclude the dead
    {
        enemies[i].update(pc.x, pc.y, 3);
    }
}

//Fish
for (int i = 0; i < sizeof(fish) / sizeof(fish[0]); i++)
{
    if (fish[i].isAlive) // exclude the dead
    {
        fish[i].update(pc.x, pc.y, 5);
    }
}
```

Figure 15 - The 'update' function implemented for different NPCs

3.4 Objects

There were several different objects created for the development of 'Treasure Quest'. All of these objects had different purposes throughout the game. For example, the 'rock' objects' main purpose was to restrict the PC's movement around the screen, and the seaweed objects were part of the environment that helped to conceal the 'key' object from the player's view, to make it harder to find it. However, there were also some objects that the player had to purposefully interact with in order to complete a level. One of such objects was the 'Key' object, which was designed so that the player would be able to pick it up and use it later on to open a door in level 2.

In order to implement all this functionality for these objects, the 'Object' class included variables like 'isActive', 'isVisible', 'x', and 'y'. The 'isActive' variable helped to determine if an object is supposed to be loaded onto the screen, and if the PC is able to interact with the object at a certain point during the game. Similarly, the 'isVisible' variable was only used for certain objects, such as the 'key' object. It was used so that these types of objects were able to be added to the PC objective after interacting with them. Finally, the 'x' and 'y' variables helped to determine the positions of the object on the screen.

Lastly, there was another class created specifically for the 'boulder' object, called 'InterObject'. This was done in order for the PC to be able to interact with the object in a slightly different way from the other objects. The difference between this class and the 'Object' class was the inclusion of variables

that allowed the dynamic change of the boulder's position and velocity attributes, allowing the PC to move the boulder from its original position.

```
if (key.isActive && pc.hasKey)
{
    key.x = 500;
    key.y = 8;
}
```

Figure 16 - The 'isActive' variable helping to determine if the key is in the PC's equipment

```
if (key.isVisible)
{
    keyRect.x = key.x;
    keyRect.y = key.y;
    keyRect.h = keyRect.w = 1;

    if (SDL_HasIntersection(&pcRect, &keyRect))
    {
        key.isVisible = false;
        key.isActive = true;
        pc.hasKey = true;
    }
}
```

Figure 17 - The 'isActive' and 'isVisible' variables working together to allow the PC to pick up the key

```
if (chest.isActive)
{
    chestRect.x = chest.x;
    chestRect.y = chest.y;
    chestRect.h = chestRect.w = 100;
    if (SDL_HasIntersection(&pcRect, &chestRect))
    {
        message.isActive = true;
    }
    else message.isActive = false;
}
```

Figure 18 - The creation of the 'message' object

```
for (int i = 0; i < sizeof(rocks) / sizeof(rocks[0]); i++)
{
    if (rocks[i].isActive)
    {
        rockRect.x = rocks[i].x;
        rockRect.y = rocks[i].y;
        rockRect.h = rockRect.w = 10;

        if (SDL_HasIntersection(&pcRect, &rockRect))
        {
            pc.xVel = -pc.xVel, pc.yVel = -pc.yVel;
        }
    }
}
```

Figure 19 - The creation of the 'rock' objects

3.5 Interface

The main user interface lets the player to control the PC using the keyboard, predominantly using the 'W', 'A', 'S', and 'D' buttons to move the character around. Additionally, the player is also able to use the 'space' button to fire an arrow to attack enemies, and destroy certain objects.

Furthermore, there is also other part of the interface that allow the player to control other parts of the game, other than the PC. For example, they can use the '1' and '2' buttons on the keyboard to control the start and end of the background music. Also, the 'escape' button on the keyboard lets the player stop the game loop, and exit the game entirely. Similarly, the 'P' button also allows the player to replay the game, if they manage to complete it the previous time. This is communicated to the player by the end screen containing a message that informs the player of that possibility.

```
case SDL_KEYDOWN:

    switch (playerInputEvent.key.keysym.sym)
    {
        case SDLK_w: pc.moveXY('u'); break;
        case SDLK_s: pc.moveXY('d'); break;
        case SDLK_a: pc.moveXY('l'); break;
        case SDLK_d: pc.moveXY('r'); break;
        case SDLK_SPACE:
        {
            bullet.fire(pc.x + 50, pc.y + 15, pc.angle + 90);
            Mix_PlayChannel(-1, arrowSound, 0); break;
        }
        case SDLK_ESCAPE: gamePlaying = false, replay = false; break;
        case SDLK_1: Mix_PlayMusic(music, -1); break;
        case SDLK_2: Mix_FadeOutMusic(2000);
        if (message.isActive)
        {
            case SDLK_e:
            {
                chestClosed.isActive = false;
                chestOpen.isActive = true;
                treasure.isVisible = true;
                if (treasure.isVisible)
                {
                    treasure.x = 350;
                    treasure.y = 275;
                }
                chestOpen.x = 300;
                chestOpen.y = 250;
            } break;
        }
        if (goalReached)
        {
            case SDLK_p:
            {
                goalReached = false;
                pc.isAlive = true;
                pc.lives = 3;
                airSupply = 61;
                currentLevel = 1;
                createLevel();
            } break;
        }
        break;
    }
}
```

Figure 20 - The 'playerInputHandler' function that implements the interface into the game

3.6 Game Management

There are two main game loops that control the general flow, and the states of the game. However, these loops also have separate functions within them that keep track of all of the changes made to the object and character variables. In turn, these functions help to keep track of the player's progress throughout the game.

On top of that, the main two lose states for 'Treasure Quest' are when the 'lives' variable for the PC, or the 'airSupply' variable reach zero. When these states become true, the 'gamePlaying' variable becomes 'false' and the game ends. However, the win state for the game is for the 'goalReached' variable to become 'true'. This can only happen at one point during the game, when the PC comes to contact with the treasure at the end of level 3.

On the other hand the main one of the other main variables that control the flow of the game, but doesn't specifically affect the game loops, is the 'currentLevel' variable. This variable helps to determine which level the player is going through, and it helps to determine which level should be loaded by the 'createLevel' function.

```
if (currentLevel == 1)
{
    if (pc.hasKey == false)
    {
        if (pc.y <= 60) pc.yVel = -pc.yVel;
    }
    if (pc.y <= 0 + SPRITE_SIZE)
    {
        currentLevel = 2;
        createLevel();
    }
}
```

Figure 31 - The conditions necessary for level 1

```
if (currentLevel == 2)
{
    if (pc.y >= SCREEN_HEIGHT - 50) pc.yVel = -pc.yVel;
    if (boulder.x >= 550 || boulder.x <= 370) boulder.used = true;
    if (boulder.used && (pc.x <= 560 && pc.x >= 460 && pc.y == 330))
    {
        currentLevel = 3;
        createLevel();
    }
}
```

Figure 22 - The conditions necessary for level 2

```
if (currentLevel == 3)
{
    if (enemiesLeft <= 0)
    {
        chestClosed.isActive = true;
        chestClosed.x = 300;
        chestClosed.y = 250;
        if (message.isActive)
        {
            message.x = chestClosed.x - 20;
            message.y = chestClosed.y - 30;
        }
        if (pc.x <= 30) pc.xVel = -pc.xVel;
    }
}
```

Figure 23 - The conditions necessary for level 3

```
if (goalReached)
{
    Mix_PlayChannel(-1, gameCompleteSound, 0);
    message.isActive = false;
    boulder.isActive = false;
    treasure.isActive = false;
    pc.isAlive = false;
    bullet.isActive = false;
    chestClosed.isActive = false;
    chestOpen.isActive = false;
    for (int i = 0; i < sizeof(rocks) / sizeof(rocks[0]); i++)
    {
        if (rocks[i].isActive) // exclude the dead
        {
            rocks[i].isActive = false;
        }
    }
}
```

Figure 24 - The conditions for reaching the goal of the game

4. Testing, Problems & Solutions

Priority	Problem	Solution	Implemented
High	The arrow doesn't fire from the correct place when the space is pressed.	Change the starting position of the arrow relative to the character, and also change the size of the arrow for better display of the sprite.	Has been implemented.
High	The level layout doesn't display properly.	This is due to the different sprite sizes of all of the objects. Therefore, the 'levels.h' file can be deleted, and the sprites can be manually placed on the screen in the 'createLevel' function, using the	Has been implemented

		'currentLevel' variable for generating different screens for the levels.	
Medium	The bullet doesn't eliminate the seaweed when in contact.	Increase the area where the 'hasIntersection' function has effect on, by increasing the height and width of the rectangular area of 'seaweedRect'.	Has been implemented
High	The character leaves the screen to appear on the other side too early.	Change the parameters for when the 'ScreenWrap' function takes applies, due to a larger size of the PC sprite than originally anticipated.	Has been implemented
Medium	Both of the seaweed and rock objects changed their sprites to 'Rock'.	This is due to the duplication of the SDL command for generating sprites, therefore removing one of the commands will help to generate the 'Seaweed' sprite for the correct object instead.	Has been implemented
High	The uneven sprite sizing prevented the proper functionality of a command that reversed the PC velocity when in contact with the rock objects.	Crop the PC sprite in an image editor, and import the new image into the 'image' folder, so that the new image is generated in the game.	Has been implemented, however it still doesn't properly work for small velocities.
Medium	The background for the next level doesn't load.	Specify that the background should load when the 'currentLevel' variable is equal to the number of the level after.	Has been implemented.
Low	The PC doesn't appear in the correct place on the screen after moving onto the next level.	Place the PC manually in the code, in the same place where the sprites for the new level are generated.	Has been implemented
Medium	The level layout doesn't reverse back to level 1 when the player is trying to go back from level 2.	Make sure that the player can't go back a level by restricting the player to the screen, so that they can't go through to the previous level from by passing through the bottom of the screen.	Has been implemented

High	The boulder got stuck to the PC upon contact.	Create a new class called 'InterObject' that is similar to the class 'Object', however containing variables for the x and y velocity.	Has been implemented
High	The boulder moved with the PC even when the PC was moving back.	Set the velocity of the boulder to be greater than the velocity of the PC, by adding to the 'pc.xVel' and 'pc.yVel' variables.	Has been implemented, however the boulder can only move to the right.
Low	The fish keep moving towards only one of the waypoints.	Placing the 'update' and 'patrol' functions in a different place in the code, and in a different order, so that the waypoints updated automatically after each fish reached their designated waypoint.	Has not been implemented, due to not being able to determine the correct place for the two functions.
Medium	The boulder doesn't disappear from the screen when moving from level 2 to level 3. However, it can be moved off the screen by the PC, so that it disappears.	The boulder's 'isActive' variable is still set to 'true' when advancing to level 3, so setting this variable to false should deactivate the boulder, and consequently it should disappear during the transition.	Has not been implemented because setting the variable to 'false' in multiple instances of the code still didn't deactivate it.
Low	The closed chest doesn't disappear after the open chest gets activated. However, it ends up hidden behind it, so it's not very noticeable.	The chest's 'isActive' variable is still set to 'true' even after activating the other chest, so setting this variable at that instance should deactivate the closed chest.	Has not been implemented because setting the variable to 'false' in multiple instances of the code still didn't deactivate it.
High	When the PC intersects with the enemy, the PC continuously loses lives.	Deactivate the enemy after the PC intersect it for the first time, consequently killing it. At the same time preventing it from dealing further damage.	Has been implemented
High	The chest doesn't appear when all of the enemies are killed.	Move the part of the code that creates the chest from the 'createLevel' function into the 'checkGameStates' function because the 'createLevel' function is only called at the beginning of each level, which made it not	Has been implemented

		update the parameters for the chest until the end of the level.	
Medium	The PC crashes into the enemies after moving to level 3.	Set the velocity of the PC to zero at the start of level 3, so that it starts off stationary, and doesn't have the chance to collide with the enemies until the player makes the character move again.	Has been implemented.
High	The chest doesn't open when 'e' is pressed.	Create separate objects for the closed and open chests, so that the second one can be activated when 'e' is pressed. This is because there is an issue with different sprites being loaded for the same object within the same level.	Has been implemented.
High	Hiding the GUI after the end screen appears prevents the whole game from advancing past the welcome screen.	Using an if statement instead of while loop when initialising the conditions for the GUI to disappear. This is because using a while loop created an infinite loop, which prevented the rest of the code to execute.	Has been implemented
Medium	The lives and air left are not reset after restarting the game.	Setting the lives to be equal to 3, and setting the air left to maximum again after each play-through.	Has been implemented for the lives, but not for the air left, because it's based off the 'SDL_GetTicks' function, which starts counting from the time when the program was first executed.
Low	The start screen appear when the PC dies, which doesn't make sense in the context.	Make a title screen that can also work for an end of game screen, so that it's not confusing when it appears at the end.	Has been implemented
High	The created image for the start screen didn't size	Change the image file from a jpg to a png, due to different file types rendering in different ways and with different resolutions.	Has been implemented

	properly with the game window.		
High	Starting the music at the beginning of the game prevents the game from moving forward.	Create buttons that allow for turning the music on and off instead, since starting the music in the main function straight away creates problems.	Has been implemented
Low	The sound of the arrow firing doesn't play when the space is pressed.	Remove the 'break' statement after the 'fire' function, since it doesn't let the next line that included the sound to be executed.	Has been implemented
Medium	The PC dies shortly after, or even instantly, after replaying the game.	This is due to the air supply variable not resetting at the start of a new play through. Since the variable for air left is based on the 'SDL_GetTicks' function, which starts counting from the time when the program was first executed, was not able to find a solution for an alternative solution.	Has not been implemented, needs more research
Low	The game over sound plays at the beginning of the game.	This issue is connected to the Boolean variables being set to 'true' and 'false' values in different places within the code, especially the 'replay' and 'gamePlaying' variables. This is because the game over sound is playing when the 'gamePlaying' variable is set to 'true'.	Has not been implemented. Couldn't find the part of the code with the issue.

5. Critical Review

Altogether, considering all of the criteria for designing, developing and implementing a 2D game using the C++ programming language, 'Treasure Quest' can be considered an overall success. However, there are always improvements that can be made to construct even more suitable and fitted programs for running computer games.

Firstly, the game is considered an encompassing success because it's in a playable state for the end user. 'Treasure Quest' is a game that is simple to play using a keyboard, with a sophisticated control system. Therefore, this suggests that this game can be enjoyed by anyone that decides to play it, since it doesn't include any major problems that would prevent the user from playing it.

Furthermore, it could be argued that the game was designed in such a way to make its difficulty allow for anyone to successfully complete the game. 'Treasure Quest' is a simple game that doesn't require previous experience with computer games to allow the user to accomplish the main goal of the game.

Moreover, looking from the graphic design perspective 'Treasure Quest' was made using free open source sprites and background obtained from the internet. However, despite this fact, the game is still pleasing to look at and all the objects seem to belong together nicely. Also, based on colour theory, none of the hues and shades in the game clash with each other, which makes the game look more professional and polished.

On the other hand there are some improvements that can be made to both the design and overall implementation of 'Treasure Quest'. For example, it would be beneficial to make it easier for the player to know what their next objective is by providing them with more visual clues, so that they can be certain that they are completing the game in the intended way.

Secondly, one of the major improvements that should be implemented for the game is the inclusion of more variables that are specific for controlling the flow of the game, so that it's possible to replay the game, even after the player completes the game in the previous round. Since, there are currently some bugs in the game that prevent the player from doing so, due to the fact that some of the variables influence the 'gamePlaying' variable, that evaluates to 'false' too early. This gives the game an unfinished feel, and so makes it less desirable for the player.

As the last point for the improvements for the game, it is an issue that is not seen from the player's point of view, but more from a developer's stand point. The big disadvantage of the code for 'Treasure Quest' was the fact that it is not organised particularly well. This is because there are some small pieces of code scattered around the whole program that help to fix some problems with the code temporarily, however shouldn't be placed there by themselves in the final implementation of the game. This makes the code harder to follow and edit in the future by someone else, and so the game wouldn't be extremely difficult to edit by another programmer.

6. Conclusion

On a final note, the design, development, and implementation of 'Treasure Quest' were completed to an acceptable standard. In the end, the game was able to be fully completed and implemented to a playable state, with being enjoyable and engaging to the player. However, there were certain drawbacks to the game's development that could be potentially improved in the future.

Overall, the main key concepts that were learned from the development of a game using the C++ programming language and the SDL library were that it's extremely important to have a clear plan of action before starting to implement the game. This is because starting the development phase without a specific list of functions and variables necessary, can in turn lead to complications down the line of development. For example, it could lead to random snippets of code being placed in the wrong places within the code, which then make it very difficult to debug the code when problems arise later on in the implementation phase.

Furthermore, it could be more beneficial to pay more consideration to the memory usage of the program when implementing it, in the future. For instance, using less sprites and backgrounds, or sprites that use up less memory would be more memory efficient, and it should not affect the player experience significantly.

Bibliography and References

3D-Ace Studio. 2020. *2D Game Art Styles: The Ultimate Guide* | 3D-Ace Studio. [online] Available at: <<https://3d-ace.com/blog/2d-game-art-styles-the-ultimate-guide/>> [Accessed 10 May 2022].

Baldwin, S., 2017. *Types of Fun 2D Games*. [online] It Still Works. Available at: <<https://itstillworks.com/12342409/types-of-fun-2d-games>> [Accessed 10 May 2022].






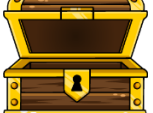

Jordan, J., 2021. *What Are the Different Types of 2D Games?* | NarraSoft. [online] NarraSoft. Available at: <<https://narrasoft.com/what-are-the-different-types-of-2d-games/>> [Accessed 10 May 2022].


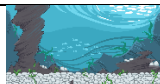
Schaefer, K., 2015. *5. Designing with Purpose*. [online] Medium. Available at: <<https://medium.com/experimenting-on-purpose/5-designing-with-purpose-9ab8c1e42a13>> [Accessed 10 May 2022].

Libsdl.org. 2022. *Simple DirectMedia Layer - Homepage*. [online] Available at: <<https://www.libsdl.org/>> [Accessed 10 May 2022].

Square Co., Ltd., 1987. *Final Fantasy Screenshot*. [image] Available at: <<https://www.gameinformer.com/2021/07/01/final-fantasy-1-2-and-3-pixel-remasters-arrive-later-this-month>> [Accessed 10 May 2022].

Graphical Assets

Image	name	Used for	Sourced from
	arrow	Arrow projectile	https://opengameart.org/content/arrow-1
	Battlefield	Background for Level 3	https://craftpix.net/freebies/free-elven-land-game-battle-backgrounds/
	boulder	Boulder object, blocking the entrance in Level 2	https://craftpix.net/freebies/free-tropical-medieval-city-2d-tileset/?num=1&count=8&sq=tropical&pos=3
	Building	Building object from Level 2	https://craftpix.net/freebies/free-tropical-medieval-city-2d-tileset/?num=1&count=8&sq=tropical&pos=3
	ChestClosed	Closed chest object from Level 3	https://craftpix.net/freebies/free-underwater-world-2d-game-objects/?num=1&count=10&sq=underwater&pos=5
	ChestOpen	Open chest object from Level 3	https://craftpix.net/freebies/free-underwater-world-2d-game-objects/?num=1&count=10&sq=underwater&pos=5
	Fish	Fish objects from Level 1	https://opengameart.org/content/aqua-fish

	gameEnd	Background for the end of the game if the player succeeds	https://opengameart.org/content/2d-background-underwater + edited by Julia Kolano
	Key	Key object from Level 1	https://opengameart.org/content/the-secret-key
	Orc	Enemy NPCs from Level 3	https://opengameart.org/content/10-basic-rpg-enemies-foxblinks-upscale
	PC_Idle	Main playable character	https://craftpix.net/freebies/free-underwater-world-2d-game-objects/?num=1&count=10&sq=underwater&pos=5
	Rock	Rock objects that surround the levels in Level 1 and 3?	https://craftpix.net/freebies/free-tropical-medieval-city-2d-tileset/?num=1&count=8&sq=tropical&pos=3
	rock2	Rock objects that surround the level in Level 2	
	Scroll	Message object appearing when the player gets close to a chest in Level 3	https://opengameart.org/content/pixel-parchment-ui-kit
	Seaweed	Seaweed objects from Level 1	https://craftpix.net/freebies/free-underwater-world-2d-game-objects/?num=1&count=10&sq=underwater&pos=5
	startScreen	Background for the start and end of the game if the player dies.	https://opengameart.org/content/2d-background-underwater + edited by Julia Kolano
	Treasure	Treasure object from Level 3	https://craftpix.net/freebies/free-underwater-world-2d-game-objects/?num=1&count=10&sq=underwater&pos=5
	Underwater	Background for Level 1	https://craftpix.net/freebies/free-elven-land-game-battle-backgrounds/
	Village	Background for Level 2	https://craftpix.net/freebies/free-elven-land-game-battle-backgrounds/

Audio Assets

Sound	Used for	File & Source
Background Music	Background music for all of the levels	Woodland Fantasy.wav https://opengameart.org/content/woodland-fantasy
Arrow and bow	Bullet fire function	Shoot.wav

		https://opengameart.org/content/bow-arrow-shot
Pained grunt	PC damage	Damage.wav https://opengameart.org/content/playersounds-1-by-emopreben
Monster snarl	Enemy death	Goblin Death.wav https://opengameart.org/content/goblin-death
Sparkling sound effect	Treasure and Key pickup	Accept.wav https://opengameart.org/content/ui-accept-or-forward
Happy high-pitched melody	Completion of the game state.	Round_end.wav https://opengameart.org/content/oldschool-win-and-die-jump-and-run-sounds
Sad low-pitched melody	Game over state after the character dies	Death.wav https://opengameart.org/content/oldschool-win-and-die-jump-and-run-sounds

Appendices

How to play

- The player is able to turn the music on and off throughout the game by pressing the 1 and 2 buttons on the keyboard respectively.
- After the player appears underwater at the start of the game, they need to find a key that is hidden behind the seaweed (The seaweed can be destroyed using the arrow by pressing space)
- At the same time they need to make sure to avoid the swimming fish because they deal damage when touched.
- Then, the player needs to advance to the next level by going through the gap at the top of the screen.
- They are then transported into the next level with a boulder blocking the entrance to a building, therefore they need to move the boulder to advance to the next level.
- After, the player is transported into a room with enemies that will try and attack them.
- So the player needs to avoid contact with the enemies, and attack them with arrows. (It's possible to move from one end of the screen to the other by going off the edge of the screen.)
- The player needs to get rid of all the enemies, in order for a chest to appear in the middle of the room.
- The player then needs to get close to the chest and press the 'E' key to open it.
- After the chest is opened, the player can go over the treasure inside of the chest to collect it. Collecting the treasure is the winning condition of the game.