Game Development

Hollow Knight

(Group: OOPS)
Shaheer Kamal
Omer Rastgar

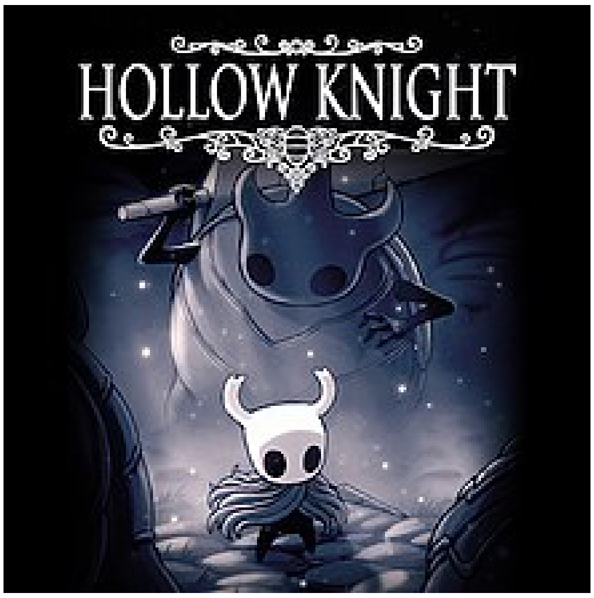
Making a game in C++

Object Oriented Programming
Habib University
12/25/20

Contents

Chapter 1

Introduction



1.1 Brief overview

We are replicating a PC game known as "Hollow Knight" as part of our OOP final project.The specific level which we are trying to code is "City of Tears" from the game .

1.2 Objective

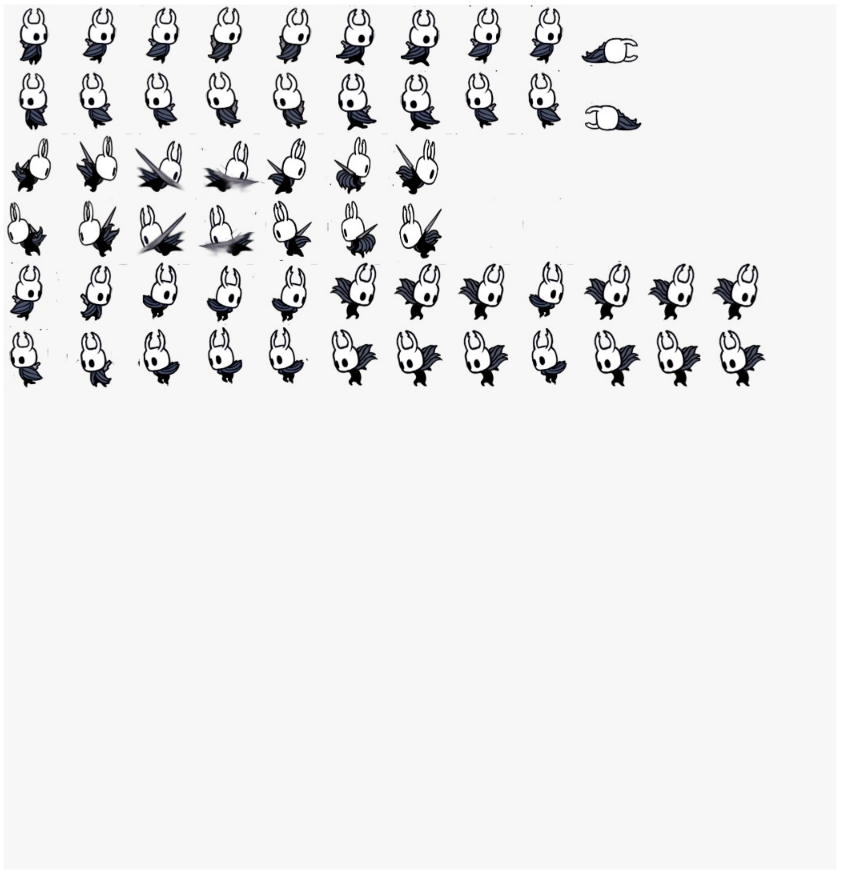
To use object oriented programming concepts in C++ to replicate a famous PC game .

1.3 Notes

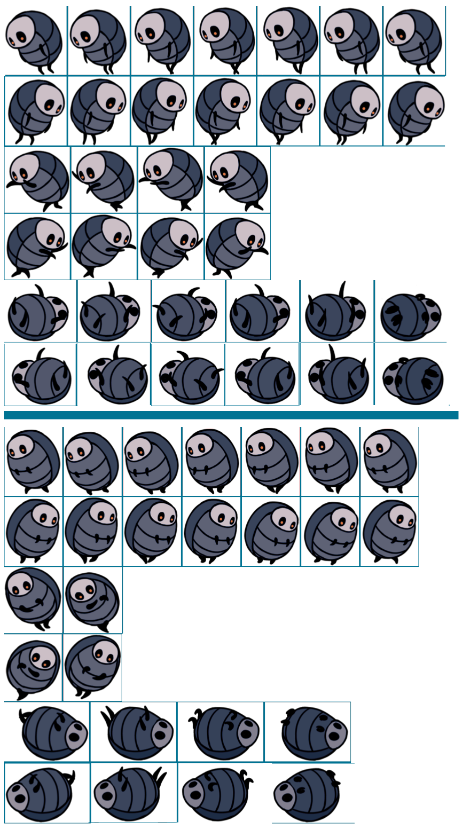
1.3.1 Game look:

it is a plateformer game hence it has objective that you need to complete, enemies to fight and a boss to destroy.

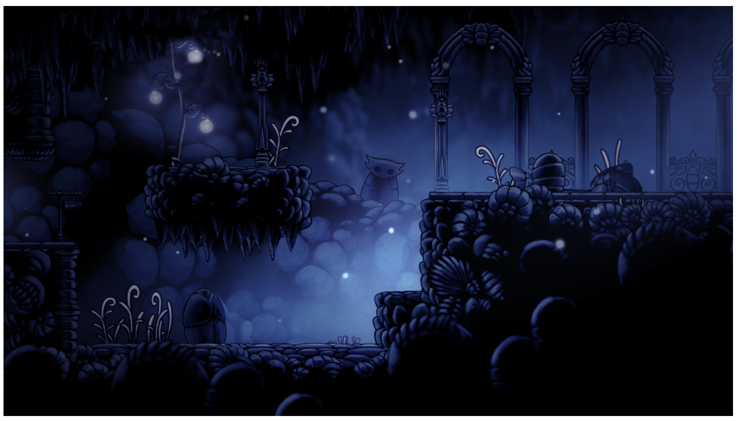
Sprite Sheet:



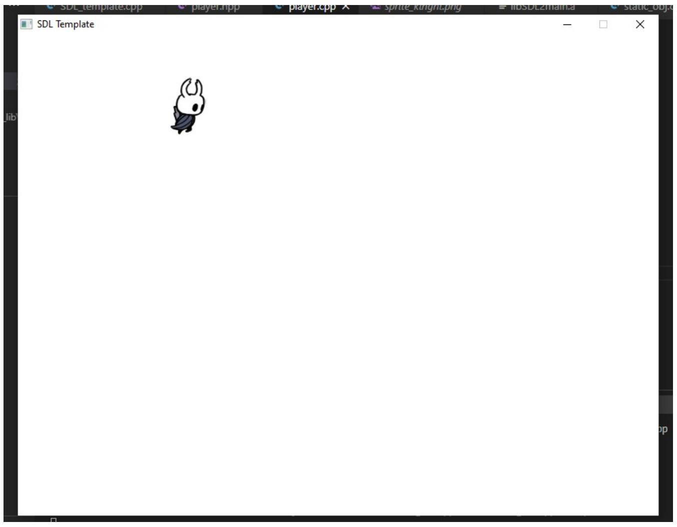
Sprite Sheet for enemy:



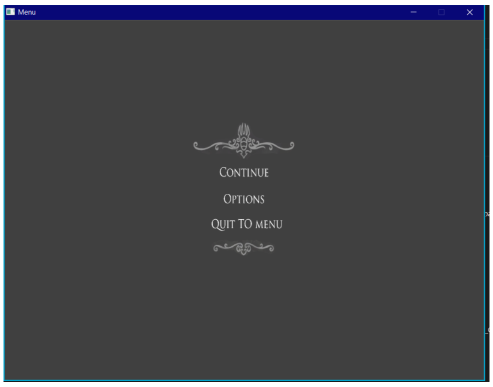
background image



Movement Implementation in SDL:



menu (transparent screen) in SDL:



1.3.2 Enemies:

Husk Dandy
Husk Hornhead
Husk Sentry
Lance Sentry
Leaping Husk
Wandering Husk

1.3.3 Weapons:

Needle Sword
Nengeful Spirit
Dash Slash

1.3.4 Attacks:

Two attacks

1.3.5 Health:

Three souls
Healing abilities

1.3.6 Places:

1.3.7 Sound tract:

”Forbidden cross roads”

Chapter 2

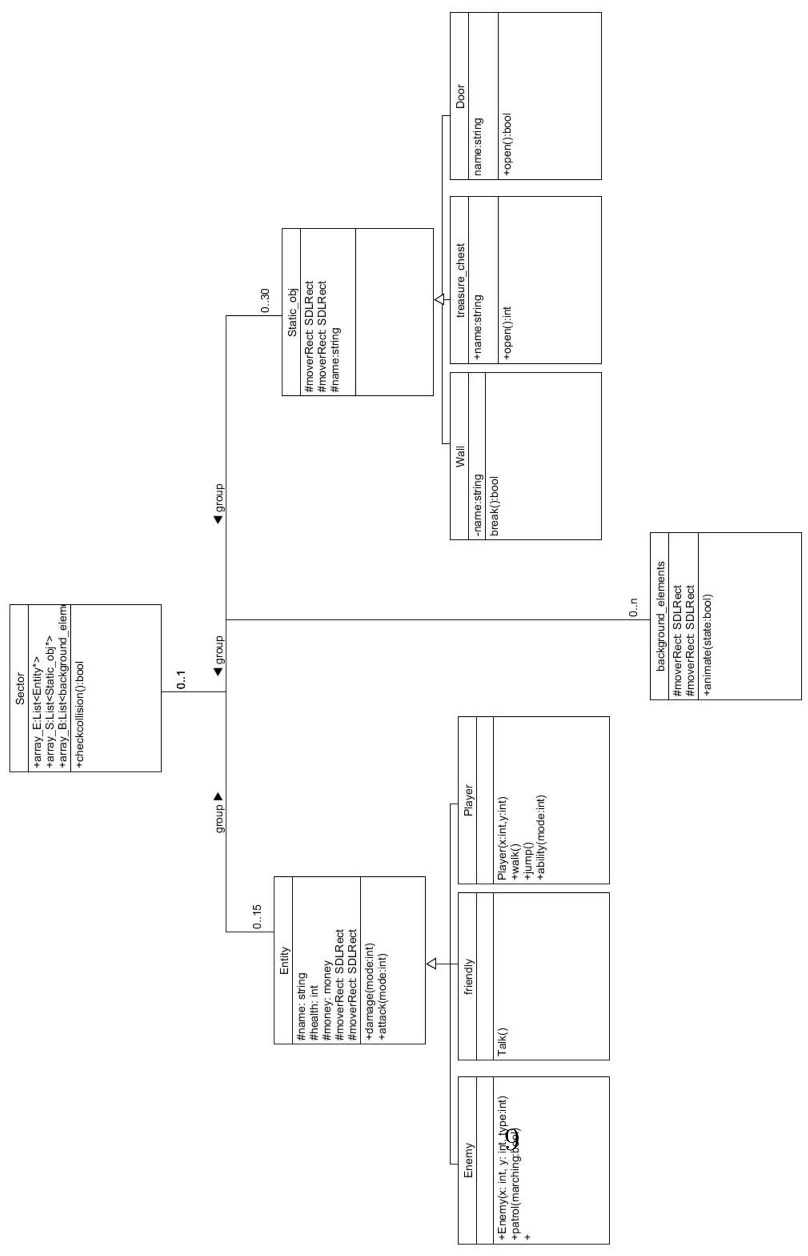
Contribution

2.1 Details

- Git Kraken to collaborate and save each persons contribution `https://github.com/or05554/Hollow-Knight-cpp/tree/main/scripts`
- using Glo board to assign work `https://app.gitkraken.com/glo/board/YFg6jyrR4wARI47Z`
- using Git Kraken timelines to keep track of time`https://timelines.gitkraken.com/timeline/cabc6398bb7e4beeb6eb03ebbea1de01`

Chapter 3

UML diagram



3.1 Explanation:

The Button and the sound manager class have an associative relation with the Menu class.The slider and the toggle classes inherit from the Button class while the sound class inherits from the sound manager class.

Since we are trying to replicate the game we need to have all the features that are present in Hollow Knight hence we have sliders for music control and toggle switches for other effects

The sound class is there to have all mix chunk and mix music ptr to files for all sound effects and background music

The music manager will be managing all the objects created, hence if we go to the options in the menu we will see that we can change the sound volume and turn on and off sound effects

The Entity,Static obj and the background elements classes have an associative relation between with the sector class.

In the static class we have doors that can be opened when a particular task is completed, the wall act as barriers and the treasure chest will provide you will money that you can use to buy upgrades

The sector class will contain all list for a specific sector, meaning since the level is very large we will need to keep the objects closer to the player in active mode, hence they will be always checking collision and strikes so that if the player comes closer to the treasure chest the two Rects will intersect and then if the attack button is pressed the treasure chest will open. if it was a wall then the player would not be able to move ahead. hence sector class will manage all major parts, including the animation in the background

The Enemy, friendly and the Player classes inherit from the Entity class while the Wall,treasure chest and the Door classes inherit from the Static obj class.

since Enemy player and friendlies have similar properties they are sharing major properties through inheritance, the enemy has to locate the player and when it comes closer it has to attack

To sum up, there will be many different sectors (about 5) which will contain linked list that will be checking collision for all objects and different animations that should be played for a specific action (death animation when player kills an enemy). As the player moves, the image will adjust accordingly and then at a point the image will not move and you will need to go through a door or passage if you have to explore further. we can pause at any time during this the linked list for loops will be disabled hence no animation or collision detection. you can also navigate to the main menu.

3.2 Design Patterns

3.2.1 Creational design pattern

Factory design pattern

Objects of certain classes are created only as per the provided input.Different from the naive approach in which objects of all the classes in the code are created.

Singleton design pattern

Only one instance of a particular class can be created.

Builder design pattern

Construction process of complex object is separated from its representation so that we can use the same construction process for different representations.A complex object will be made step by step and the final product will be that object. Example can be the construction of a PC from its components like motherboard , monitor etc.

3.2.2 Structural design patterns

Adapter design pattern

An adapter class is made such that to link to non compatible classes/ components in the program.Eg: the adapter class will take an input from a component (component 1) and give it to another non compatible component (component 2) to make them compatible,It is also used when we know beforehand that a particular part of the code can change in future.

3.2.3 Decorator design pattern

This design pattern attaches additional responsibility to an object dynamically. Provides flexible alternative to sub-classing for extending functionality.

Facade design pattern

It hides the internal complexity of a system and provides a simple interface. For example if in our conventional design pattern if we need to execute five functions for a task then in the facade design pattern we can just make a new facade class and make just two functions inside it which will execute the previous five functions

3.2.4 Behavioral design patterns

Observer design pattern

It defines the one to many relationship between objects so that when one object changes its state so all of its dependant objects get notified and change their states accordingly.Example can be the case when we get notification triggers we get when we subscribe a channel on Youtube or like a page on Facebook etc.

Command design pattern

Used when we need to implement a command type algorithm inside our program. Separate classes will be created for different commands and then an invoker class will be created which will execute all these commands. Example of usage for this design pattern can be when our application supports undo, redo or transactional functionality or writing and deleting form a database.

Strategy design pattern

Used when we have different algorithms and need to select one at run-time. For example it can be used when some messages our coming to a system and at run-time it will decide if the messages will be written in files, queues or databases

Chain of responsibility design pattern

In this design pattern we pass some requests to an objects and if an object can handle it then it will handle it otherwise it will pass to the next objects. Example can be of an ATM machine in which when we request a certain amount of a currency (eg : 2000 dollars) so it checks if that amount exists with multiple combinations (such as 100,200,500 or 1000 dollars etc and if say that amount exists with 1000 dollars so it will return two notes of 1000 dollars.)