**FINAL PROJECT REPORT**

**Teacher:**

**Nguyễn Văn Vũ**

**Trần Duy Quang**

**Lê Hoàng Dũng**

**STUDENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| S/N | Name | Student ID | Class |
| 1 | Nguyễn Trung Tín | 19127582 | 19CLC6 |
| 2 | Bùi Huỳnh Trung Nam | 19127478 | 19CLC6 |
| 3 | Ngô Trường Tuyển | 19127622 | 19CLC6 |

**Contents**

[**I/ TOPIC** 3](#_Toc51664699)

[**II/ TECHNIQUES ARE USED** 3](#_Toc51664700)

[**III/ NEW FUNCTION** 5](#_Toc51664701)

[**IV/ BRIEF EXPLANATION** 5](#_Toc51664702)

[**1/ SCENE GROUP (MANY SCENE CLASSES)** 5](#_Toc51664703)

[**2/CLASS SCENEGAME** 6](#_Toc51664704)

[**3/CLASS SCENESTATEMACHINE:** 7](#_Toc51664705)

[**4/CLASS GAME** 7](#_Toc51664706)

[**5/ OBJECT GROUP** 8](#_Toc51664707)

[**6/ ITEM GROUP** 8](#_Toc51664708)

[**7/CLASS MENU (DYNAMIC MENU)** 9](#_Toc51664709)

[**8/ CLASS PLAYER** 9](#_Toc51664710)

[**9/ CONSOLE** 9](#_Toc51664711)

[**V/ GAME PLAY** 9](#_Toc51664712)

|  |  |  |  |
| --- | --- | --- | --- |
| **Order** | **Student's name** | **Function description** | **Estimated hours** |
| 1 | Nguyễn Trung Tín | Add background music | 4 |
| 2 | Ngô Trường Tuyển | Create classic maps | 6 |
| 3 | Bùi Huỳnh Trung Nam | Create modern maps | 6 |
| 4 | Nguyễn Trung Tín, Bùi Huỳnh Trung Nam, Ngô Trường Tuyển | Build store to sell accessories | 30 |
| 5 | Nguyễn Trung Tín | Create highest score board | 3 |
| 6 | Nguyễn Trung Tín, Bùi Huỳnh Trung Nam, Ngô Trường Tuyển | Build level(easy, medium, hard) | 15 of each members |
| 7 | Nguyễn Trung Tín, Bùi Huỳnh Trung Nam, Ngô Trường Tuyển | Build mode with two players. | 30 of each members |

**Frist of all, we thank for support from teachers when we have bugs, problems or anything relating this project.**

# **I/TOPIC**

We make a decision on snake game because we think absolutely a game project has more attraction than another project, we find interesting in doing this topic and having more choices to update or upgrade also add new several features. Partly, we think its purpose is mainly OOP which we are studying in this term so we can deepen in old knowledge or maybe can know some new techniques.

# **II/TECHNIQUES ARE USED**

In this project, we try to code fundamental contents by ourselves without referring so much in other sources code. After that we upgrade and add new several functions.

**Techniques studied are used:**

* **Object – Class**
* **Link – Associate**
* **Constructor – Destructor**
* **Overriding and Overloading**
* **Encapsulation**
* **Polymorphism**
* **(Multi-) Inheritance**
* **STL**
* **Exception**
* **Virtual function and Pure function**

**Extension:**

**Shared\_pointer**

The shared\_ptr type is a smart pointer in the C++ standard library that is designed for scenarios in which more than one owner might have to manage the lifetime of the object in memory. After you initialize a shared\_ptr you can copy it, pass it by value in function arguments, and assign it to other shared\_ptr instances. All the instances point to the same object, and share access to one "control block" that increments and decrements the reference count whenever a new shared\_ptr is added, goes out of scope, or is reset. When the reference count reaches zero, the control block deletes the memory resource and itself.

**Design pattern**

* Abstract Factory
* State
* Singleton

**Dynamic casting (Memory Library)**

1. Dynamic\_cast
2. Dynamic\_pointer\_cast

A cast is an operator that forces one data type to be converted into another data type. In C++, **dynamic casting** is, primarily, used to safely downcast; i.e., cast a base class pointer (or reference) to a derived class pointer (or reference). It can also be used for upcasting; i.e., casting a derived class pointer (or reference) to a base class pointer (or reference).

Dynamic casting checks consistency at runtime; hence, it is slower than static cast.

**Unordered\_map (Unordered\_map library)**

unordered\_map is an associated container that stores elements formed by combination of key value and a mapped value. The key value is used to uniquely identify the element and mapped value is the content associated with the key. Both key and value can be of any type predefined or user-defined.  
Internally unordered\_map is implemented using [Hash Table](https://www.geeksforgeeks.org/hashing-set-1-introduction/), the key provided to map are hashed into indices of hash table that is why performance of data structure depends on hash function a lot but on an average the cost of **search, insert and delete** from hash table is O(1).

**Bind Function (Functional Library)**

Sometimes we need to manipulate the operation of a function according to the need, i.e changing some arguments to default etc. Predefining a function to have [default arguments](http://quiz.geeksforgeeks.org/default-arguments-c/)restricts the versatility of a function and forces us to use the default arguments and that too with similar value each time. From C++11 onwards, the introduction of bind function has made this task easier.

Bind function with the help of placeholders, helps to manipulate the position and number of values to be used by the function and modifies the function according to the desired output.

**Performing Operations on console**

We define some functions to perform operation on console as go to one coordinate, change color of the text and background or hidden pointer ,...

# **III/NEW FUNCTION**

1. Created more new maps
2. Multi-Players
3. Dynamic Menu
4. Save Game
5. Load game
6. Pause Game – Continue
7. Player’s Score
8. Score Board
9. Level (pass to next round)
10. Background Music

# **IV/ BRIEF EXPLANATION**

## **1/SCENE GROUP (MANY SCENE CLASSES)**

We create an abstract class named Scene has some functions which *will be overriding* in sub-classes

In here, let say a sub-class inherited from this abstract class is one a **scene** because it has *specific attributes and purpose* performing certain screen on console.

We have some classes inherit form this class:

|  |  |
| --- | --- |
| **Name of class** | **Usage** |
| class SceneMainMenu : public Scene | Give you first menu to choose some functions of this game |
| class SceneGame : public Scene | **We will explain this class below** |
| class SceneGameOver : public Scene | Show game over screen and play again menu |
| class ScenePause : public Scene | Show pause-continue screen menu |
| class SceneSaveGame : public Scene | Save current map, snake, food, score,.. |
| class SceneLoadGame : public Scene | Continue play game which is saved |
| class SceneShop : public Scene | Show shop to buy somethings |
| class SceneScoreBoard : public Scene | Show score board of top 10 player having highest score |

**This abstract class (Class Scene) has some functions and its usage:**

|  |  |
| --- | --- |
| **Name of function** | **Usage** |
| virtual void OnCreate() = 0 | Called when scene is created (just called once at Class Game) |
| virtual void OnDestroy() = 0 | Called when scene is not necessary (The Game will be not called again this scene) |
| virtual void OnActivate() | Called whenever a scene is transitioned into. Can be called many times in a typical game cycle. |
| virtual void OnDeactivate() | Called whenever current scene has to change to other scenes. Can be called many times in a typical game cycle |
| **4 functions below perform action of scene (we call it is action function)** | |
| virtual void ProcessInput() | Read signals from keyboard |
| virtual void Update() | Write data base on ProcessInput function |
| virtual void LateUpdate() | update information and ready to show on console screen |
| virtual void Draw() | show unity on console screen |

* Those functions below can be overridden as necessary in our scenes

For example:

We are having scene A and we change to scene B, functions will be called in the order:

Scene A: OnDeactive()

Scene B: OnActive() → ProcessInput() → Update() → LateUpdate() → Draw()

In each class, we use unordored\_map to hold scenes which this class can access to change (use SceneStateMachine which will be **explained below**)

## **2/CLASS SCENEGAME**

This class is important the most in Group Scene. It creates game for playing

**In attributes:**

|  |  |
| --- | --- |
| **Variable** | **Usage** |
| std::shared\_ptr<Snake> \_snake  std::shared\_ptr<Fruit> \_fruit  std::shared\_ptr<Gate> \_gate  (this class will be explained in next part) | we use shared\_ptr for those pointers (usage of this technique is explained in II). In here we can create object snake, fruit, gate (use for pass round) to cater to this game **(play 1 or 2 people )** |
| std::vector<std::string> \_maps | Use to save name of maps |
| vector<std::shared\_ptr<Object>> objects  (this class will be explained in next part) | Use to save all objects of this game to manage easier |
| std::shared\_ptr<ScenePause> \_pauseScene  (one of classes above) | use to save current screen to ready change to PauseScene if player want to pause |
| Like above classes, this class has unordored\_map to hold several scenes | |
| Besides it, we have variables to hold current round and last round you play and somethings relating like width, height,... | |

**In operations:**

|  |  |
| --- | --- |
| **Name of function** | **Usage** |
| void loadMap() | We use files so save maps so we have a function to load it, in here we use exception to check error of files |
| std::shared\_ptr<Object>  addObject(ObjectType type, int x, int y) | Use this function to save object in vector object above |
| void deleteSnakeSegment(int x, int y);  void deleteFruit(int x, int y); | use to make animation of game |
| we have several **set/get** functions to access data or Draw function , title function,… | |
| In addition to have some functions above (overriding)  (we use functions of class snake in lateUpdate to check life or dead for snack,score and control snake. | |

## **3/CLASS SCENESTATEMACHINE:**

To can *save or access* all scene above and *hold* state which tell we current scene (curScene variable) and which scene we can change into, we create class named SceneStateMachine

We *set SceneStateMachine in each class* in Scene Group (**reference variable in each Class Scene**)

To save scene, we use unordored\_map, each map will be numbered with one integer, this work help we find scene and active this scene easier.

* **Noted:** in Class SceneGame, we use unordored\_map to save name of maps and integers, and in here, we use those integers and maps to save in unordored\_map.

We have 4 functions with the same name with **action function** in Scene Group. *In each function, we call them same name function of each scene.*

For example: (curScene is prepresent for current scene)

void SceneStateMachine::Update(){

if (curScene){

curScene->Update();

}

}

We have functions named **SwitchTo** using to change scene to other scenes

Beside it, we a vector with Player type (this class will be explained below) and addPlayer function to manage player because this game can be played by 1 or 2 player.

In here, we use **finite-state machine** which is one of **State Design Pattern** in this part

(link:<http://thatgamesguy.co.uk/cpp-game-dev-6/?fbclid=IwAR190qUkIOm94H1Aa1Z8-EeehRORXUHJZpYSqxO5W1suzkltRpcsxxcM118>)

## **4/CLASS GAME**

In this class, we *create object from scene group*, each scene will be *created and add* to SceneStateMachine (SceneStateMachine is one of attributes of Game Class, so we can know current scene to use its function or can change current scene to new scene and vice versa)

Now Class Game will has 4 functions with same name with **action function** (like 4 function in SceneStatMachine) in Scene Group, *in each function, SceneStateMachine will call function with the same name*. It like:

(**Class Game**) (**Class SceneStateMachine**) (**Scene Group**)

Draw() → Draw() → Draw()

In addition, Class Game create a circle loop like this:

If we **change current scene** to other scene, **new scene will be repeated** **this cycle** until we end this game.

## **5/OBJECT GROUP**

This class has attributes which is used to save coordinate x and y of each point displayed on console screen.

It also hold width and height of map.

So we have some functions to set and get data from this class and have one pure virtual function is paint which will overriding in each class inherited from it.

**CLASS SNAKE, FRUIT, SNAKESEGMENT, GATE, WALL**

Those class are inherited from Class Object.

|  |  |
| --- | --- |
| **Name of class** | **Usage** |
| class SnakeSegment : public Object | This class is just inherited from Class Object, it prepresent for cells of snack |
| class Snake : public Object | One of attributes which is very important is a deque of SnakeSegment type which is used to save all cell of snake  This class also has other attributes use to check for life or dead of snake and provide function which use to control the snake |
| class Fruit : public Object | Use to create fruit |
| class Gift : public Object | Use to create gift (random item) |
| class Wall : public Object | Use to create wall of this game |
| class Gate : public Object | Use to created gate for this game |

## **6/ITEM GROUP**

During playing process, beside fruit we have some random items which support for snake

So We create a abstract class named Item have only one pure function is operate function.

Those class are inherited from Class Item.

|  |  |
| --- | --- |
| **Name of class** | **Usage** |
| class Rocket : public Item | Increse speed of snake |
| class ThroughWall : public Item | Make snake can go through wall without dead |
| class X2Point : public Item | Get more scores |

## **7/CLASS MENU (DYNAMIC MENU)**

This class is difference with Class SceneMainMenu)

In this class, we will *create menu* from parameters (std∷vector) from outside class, and we can use arrow keys to choose which we want, its function will *return a integer number* for the option we choose

For Example: with vector <string> ex = {“abc”, “cba”} parameter, we can create a menu with 2 lines are “abc” and “cba”

We use std∷bind to pass parameters into this class.

In addition, this class create a logo and border,… for this game.

## **8/CLASS PLAYER**

Use to save current score of each map and total score using to buy somethings in shop (including score of 1 or 2 player).

This class provide some functions that use to add score, reset score or show score,...

## **9/CONSOLE**

Providing functions use to perform with console like go to one coordinate, change color of text or hidden pointer,…

# **10/MAIN.CPP**

In here, we create object form Class Game, and start loop play game, it look like:

Game game;

while (game.IsRunning()) {

game.ProcessInput();

game.Update();

game.LateUpdate();

game.Draw();

}

# **V/ GAME PLAY**

**Purpose:**

You have a mission that is to try to get score the most as you can.

In each map, you have to get required score to pass and go to next map.

When snake eat fruit, its length will be increased.

You can buy somethings you like in shop by score which you get when you eat fruit.

**Rule of this game:**

Use arrow keys to move in menu and choose by enter key.

Use **ASWD** to control snake:

1. W : Go Up
2. S: Go Down
3. A: Go Left
4. D: Go Right

If you are in multi-player, player 2 will control snake by arrow key:

1. Up Arrow : Go Up
2. Down Arrow: Go Down
3. Left Arrow: Go Left
4. Right Arrow: Go Right

During playing process, if we eat a random items (?) we can use 1,2,3 keys to use those items

Press ESC to pause game.

(Reference source: cplusplus.com, stackoverflow.com,github,youtube.com,…)