Bubble Trouble - Group 4

**Group members: Asif, Darshan, Navin and Sighvatur**

**Aim: To create a bubble trouble game using C++.**

**Recap on Cycle 1 implementation:**

In this iteration, the following game dynamics are implemented.

1. Once the game is started, a player is able to move either left or right using “A”, “D” keys respectively.
2. “Spacebar” key is used to shoot the spike.
3. A Spike can be shot one at a time, until it either hits the ball or hits the top most point of the frame.
4. There are different sizes of ball with different colors. Also, different size of balls have different velocities.
5. Once a spike hits a ball, it splits into two smaller sized balls.
6. The new balls created attains new velocity.
7. Frame rate is set to 60 fps.

**Cycle 2 implementation:**

In this iteration, the following game dynamics are implemented.

1. Once the game is started, a timer starts.
2. A player is given three lives initially, which starts reducing when the player collides with the bubbles.
3. Scores are given when the player pops a bubble.
4. Definite bubble sizes are created.
5. Explosion sounds of bubbles are added.
6. A burst image is formed when the spike pops a bubble.
7. The game can be paused/unpaused, exited or resumed by pressing the keys “P”, “Esc” or “Enter” respectively.

**Cycle 1 Design Decisions:**

* A game can have an engine which is responsible for rendering of the game, initializing game dynamic components, updating as the game progresses and finally cleaning of the objects when the game has ended/exited. Hence we have a “GameEngine.h” which on a high level is responsible for the aforementioned specifies.
* A game has many components and may communicate with various other classes to get the required objects. So, templating of C++ comes in handy. Hence “GameObject.h” does tracking of the newly added components/objects being added to the list. This exhibits code modularity and reusability.
* Also, the components needs to be initialized and updated as per the events taking place in a game, hence having virtual functions to handle the corresponding events is wise. So, a class called “GameComponent” with the GameObject class is placed. The virtual functions are overridden in the respective game handler class/events. Most of the classes extends “GameComponent” class. This indeed reduces code duplicity and code redundancy.
* For the current implementation, the game can be played via a keyboard. The pressing of buttons forms an event. Hence a dedicated class called “KeyboardHandler” has been made to handle the key press events. Hence, when a player presses the dedicated keys, C++ knows the events and the functionality assigned to those events. Off course, SDL libraries comes in handy here.
* In the game, various movements are observed
  + A player moves either to right or left.
  + Balls created are moved randomly in the 2D plane.
  + Once a ball is shot, it splits and becomes a smaller size from the parent ball size and attains certain velocity.
  + If the balls are in the same plane as the player, if yes, the player loses life (will be implemented in the later cycles.)

Keeping track of the positions of above mentioned scenarios are important and the checks must be made in certain scenarios. Hence a dedicated class to check the collisions, is tracked under “CollisionChecks”. Now, once the checks are made, “CollisionHandler” class will handle the events like splitting the balls into two parts and assigning the respective velocity based on it’s updated size(will be less from the parent ball size).

* As mentioned earlier, player’s, spike’s and ball’s movement position and velocity needs to be calculated. “MovementHandler” class handles tracking such parameters during the course of the game.
* Textures of the player, balls and spikes has to be loaded and rendered. Hence, a separate class to handle rendering of the different texture called “TextureLoader” class.
* In order to render spike and the player, we have class called “TileHandler” class. This handler will handle events like loading the texture of player, balls and spike (calls texturehandler) and scales the respective images accordingly.

**CYCLE 2 Design Decisions**

* Different classes for players, fonts, spikes and lives are created. This will handle the update functions needed during the course of the game.
* To make the game less predictable, the bubble object should have random properties i.e velocity, position and acceleration. This is done with the “Randominterface” class.
* TTF Font Library is implemented in order to display the score and timer.
* The “bubbleobject” class creates the bubbles and randomizes their properties, in order to make the game challenging. The random values are obtained from the “Randominterface” class.
* One of the game modes has multiple stages , which can be reached by destroying all the bubbles within certain amount of time. The timer function is tracked by the “millistimer.h” class, which also has functions to stop the time and resume when the game is paused.

**Cycle 1 class explanations:**

Following are the classes involved to achieve the above-mentioned implementations.

1. Main.cpp
   1. Entry point to the game
   2. Responsible for initiating the Game engine, window size, position.
   3. Runs the main game loop, calls handleEvents, update, cleanObject and render functions of the Game.
   4. Locks the framerate at 60fps as most of the modern computers have a screen refresh rate of 60Hz.
2. GameEngine.h
3. Handles all game logic by initiating the objects, window and background, handling events, updating all objects, cleaning them and calling draw.
4. The event handlers, SDL window and objects for the player, spikes are created here.
5. The boolean to initiate the Game isrunning() is provided from this file.
6. The GameEngine starts the game and cleans up the objects when the game has ended.
7. The tile is rendered for both bubbles and the player.
8. A color array is given to change the colours of each bubbles created.
9. The update for the player movement and spikes are created here.
10. Uses functions from Collisionchecks.h to decide to destroy the spikes and bubbles.
11. Gameobject.h
12. Includes the component base class of GameComponent in order to use it for templating.
13. Has an interface to allow components to be added and accessed, using templating.
14. Iterates over each component for each update and draw call.
15. CollisionHandler.h
    1. Extends the GameComponent class
    2. In each update call it checks if the object associated with it is colliding with the playing zone, and changes its position and velocity if appropriate.
16. KeyboardHandler.h
    1. Extends the GameComponent class
    2. Keeps track of the keyboard events, updating velocity and enabling the spike object.
    3. Implements “AD” keys for left and right movement respectively.
    4. Also “left-right arrow” keys for left and right movement respectively.
    5. “Spacebar” keys for shooting the spike.
    6. Velocity flag is flipped accordingly to the movement of the player i,e, velocity is positive towards right movement and becomes negative to move towards left.
    7. SDL\_GetKeyboardState is used to determine what keys are being pressed.
17. TextureLoader.h
    1. Loads the textures of the game by making use of the SDL image library and SDL\_CreateTextureFromSurface method.
    2. Applies color to the texture, used for the bubbles.
18. MovementHandler.h
    1. Extends the GameComponent class
    2. The position and velocity of all the game object is maintained and updated here.
19. TileHandler.h
    1. Handles loading and rendering of textures using TextureLoader.
    2. If a texture is loaded, the texture is destroyed once the game has ended.
20. Collisionchecks.h
21. The boundaries of both the player and the bubbles are checked.
22. The functions check if any of the game objects collide with one another at the point or not.
23. The function collideswithrect checks between the rectangular background, spike and the player.
24. The function collideswithcircle checks between all the game objects.

**Cycle 2 class explanations:**

**Game description:**

|  |
| --- |
|  |

**Figure-1**

**Figure 1** is popped up when the user starts the game,

1. A player can use either mouse or keyboard (moving left by using left arrow). This cancel the player from entering the game.
2. By pressing “Enter” from keyboard, a player can play the game.
3. Selecting this will again can makes the player not enter the game.

------------------------------------------------------------------------------------------------------------------------------------------

|  |
| --- |
|  |

**Figure-2**

As soon as the player enters the game, **figure 2** window is seen. The window contains

* A player
* Lives
* Score
* Time

Pressing *“*Space Bar*”* will enable a player to shoot a “spike” as seen in the figure 2. As soon as the spike hits the top of the play area, it disappears.

------------------------------------------------------------------------------------------------------------------------------------------

|  |
| --- |
|  |

**Figure-3**

**Figure 3** shows that when a player hits a bubble, the bubble splits and explosion sound is heard with a small explosion image around the newly created bubbles. As a result, point is updated and spike is destroyed.

------------------------------------------------------------------------------------------------------------------------------------------

|  |
| --- |
|  |

**Figure-4**

**Figure 4** shows that when a bubble has collided with the player, one life is lost and is updated in the lives section as shown in the **figure 2.**

**Use case diagram**

|  |
| --- |
|  |

|  |
| --- |
|  |

Use case diagram has three main use cases and rest are either extended(“extend”) from a use case or part of the same use case (“Include”).

The main use cases that an actor/player can see in the Bubble Trouble system is

* Actions
* Lives
* Menu

A player ‘s action includes

* Moving left
* Moving right
* Shoot the spike
  + Hit’s a bubble
    - New bubbles are created
    - Explosion sound is heard with a small explosion image around the newly formed bubbles
    - Score is updated
* Pausing the game &
* Exiting from the game

When the game is started, a menu must be seen and a player might select if he wants to play an infinite game mode or staged mode. Also, certain power ups such as more lives, increasing time etc. would be made available to a player to collect it. These use cases would be explained in detail in cycle 3.

**Inheritance Diagrams:**

|  |
| --- |
|  |

|  |
| --- |
|  |

**Class Diagram:**

|  |
| --- |
|  |

**Division of work for cycle 3:**

|  |
| --- |
|  |