Brick Breaker Final

By Mason Caird and Benjamin Greenwood

Project Requirements:

When we began making this game, we had to implement a function to let Visual Code keep a game playing with objects. To do that, we made a *ShowConsoleCursors* function. This function lets the terminal know to become a window inside the Visual Code program.

We set up our window width and height to 20x40. We chose 20 because that should give the user enough time to move the user box and with 40 as the height to give the user a rest while the ball bounces off the walls.

From this point, we have classes to define each of our objects that are in our game. Each class has its own complexity to make the game mechanics work the way they do. We made sure to comment on everything we could in our code if there are any misunderstanding. We have acquired all requirements including vectors, pointers, dynamic memory, inheritance, and polymorphism.

Class gameObject: (parent)

This class will be the parent class to all classes in our header. In our protected class we have our COORD pos, and our velocity variables (vx, vy). We have a default constructor setting our velocity and position variables to 0. We have a virtual draw function passing in visual code visuals support, just like the *ShowConsoleCursors* function and a virtual move function().

Class stillBox: (child)

This class is responsible for the stillBoxes properties and how they are randomly generated. String stillBoxShape is protected and in our public, we defined a default constructor stillBox( ). In this constructor, we have a rand() for our pos.X and pos.Y. Outside of the constructor, we have getX( ) and getY ( ) functions and newX( ) and newY( ) functions to repeat the position if they overlap. We also have get Shape() to return the string of characters and a draw function, hit function that returns an empty string of 7 spaces.

Class userBox: (child)

This class is responsible for the userBox properties and how the box moves according to the keyboard input. We have the protected string userboxshape. In our default constructor, the position of the userbox is 17 and 18. The reason for the higher numbers is because the default window axis start from the top left unlike the bottom left. Our move ( key ) is where the input from the keyboard comes in. We have if statements stating if 75 or 77 are pressed then the userbox will move left or right. We used another set of if statements to give the userbox bounds from leaving the window.

Class Ball: (child)

This class is responsible for the ball properties and how the ball changes direction and the velocity. We have protected char ballShape\_. We have a default constructor Ball with the ball shape parameter passing through. FlipVy() will used once the ball hits the userBox. Draw(), move() and getY() will be used. The move() function keeps the ball inside the window except the bottom.

Collide():

This collide( ) function will be used to compare a blocks position with the balls. Two parameters will be passed through as COORD for ballPosition and blockPosition.

GameOver():

GameOver() will be called to check if the game is over. The pointer vector and blockshit will will be passed as a parameter. We used typeid to check to see if the gameObject is a stillBox, then we used Dynamic Casting to retrieve getShape from stillbox to make it an empty string.

Overlap():

This function reassures all stillboxes don’t overlap when they are randomly generated. We used dynamic Casting to to get the stillboxes shape.

Main();

In main we started our function with a pointer vector of gameObject. Then initialized our handles for the output on the screen for the terminal. We push backed user Box, ball and for looped the still box. We have while statement that hold if statements to determine if any two objects collided and call various function through the while loop. We use polymorphism to draw and move each of the objects. We intake the key input at the end of the while loop to start the process over again.

UML%20Update.pdf