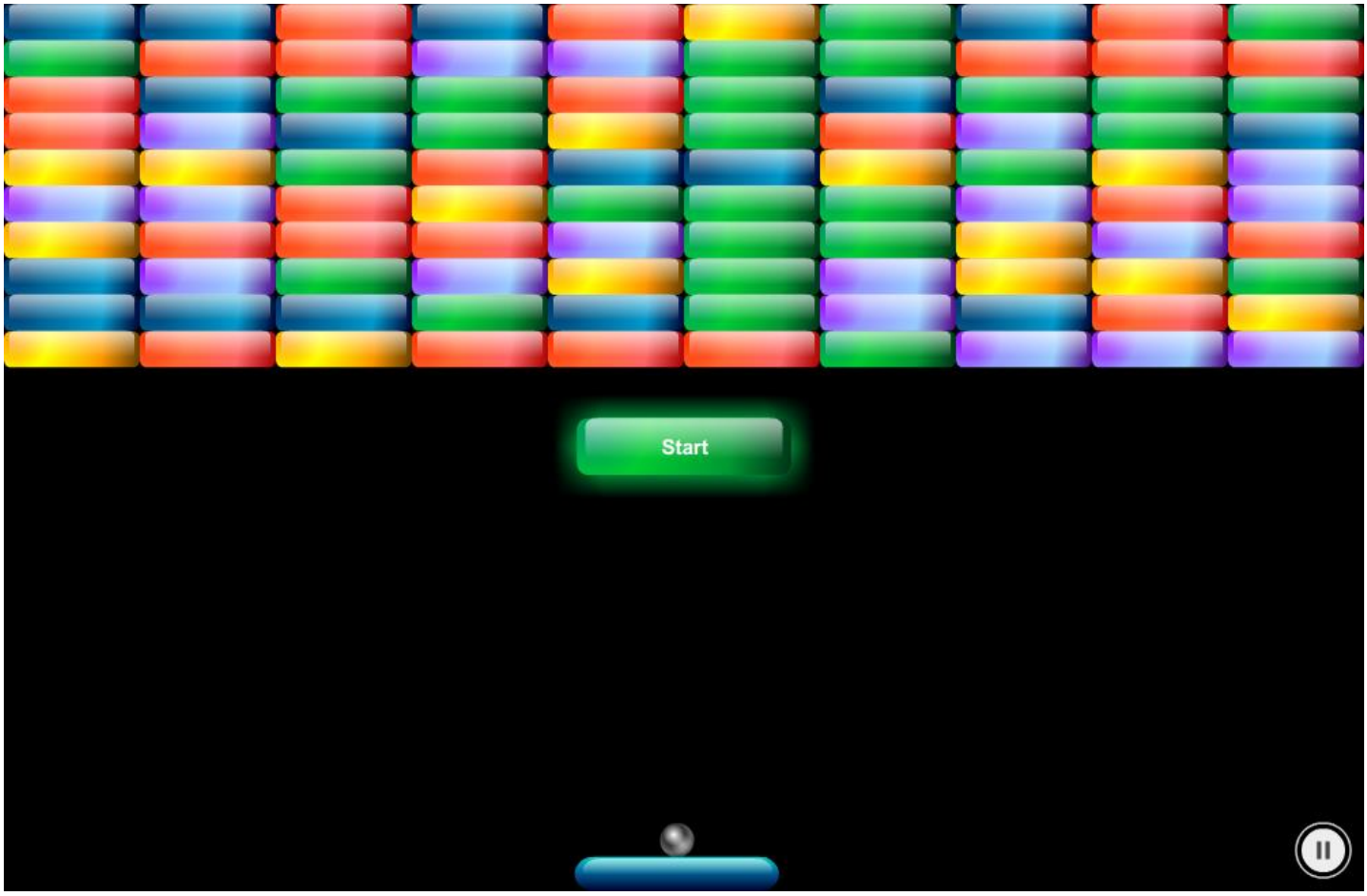
Brick Breaker – a flash game project



Computer Science Department of Crete
Multimedia Technology (hy474) – Spring 2013

Angelos Kyriakopoulos(2482) – akyriak@csd.uoc.gr

Nina Saveta(763) – jsaveta@csd.uoc.gr

The implementation of the game was made with Adobe Flash Professional CS6, in Actionscript 3 and the project is located at the repository hy474_brickbreaker_2013(inform us if you would like access to it). We will present the controls, the graphics and the source code of the game.

1 CONTENTS

2 CONTROLS & USABILITY

We will present the controls of the game and the usability that the game offers to anyone that plays it. In order to start the game the user has to press the start button.

During game play they have to move the paddle with mouse or arrow keys (left and right) to break the bricks with the ball. We offer two different ways for the movement of the paddle in order to make the game more usable. With arrow keys the game is more difficult because the speed of the paddle is slow and the user has to bounce the ball closer to the middle of the paddle, which makes the game more difficult.

The purpose of the game is to break all the bricks. To achieve that they have three chances to lose the ball that are displayed on the bottom left corner of the screen. Whenever a boll is lost they get informed by a message that flies on the screen. This message is clickable in order to continue the game.

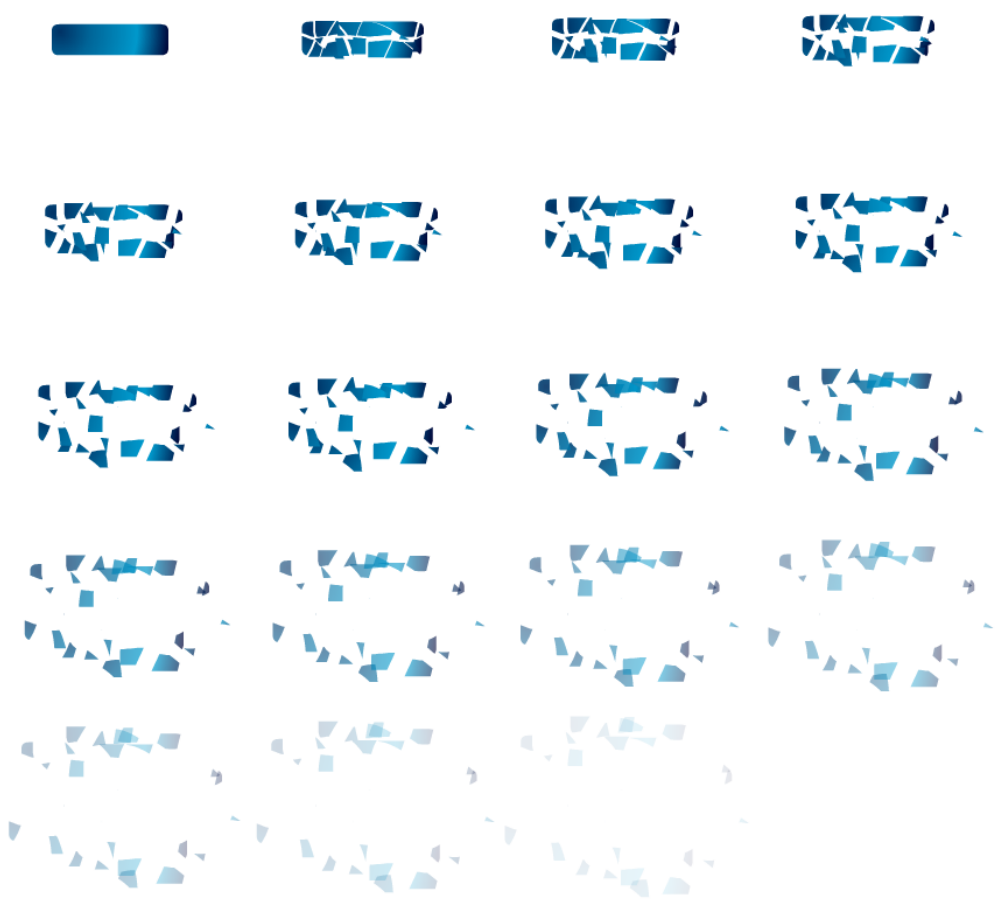
The user is able to pause and resume the game any time by pressing the “P” key on his keyboard. Also they are able to mute and play the music of the game any time by pressing the “M” key. That way the user can pause or mute the game whenever they like.

3 GRAPHICS

3.1 BRICKS

Every brick has is represented by two symbols in the flash library. The first symbol represents each brick before it breaks is a movieclip symbol and is displayed on the screen as below.

However the second symbol consists of many different graphic symbols that are displayed while the brick breaks. Every graphic symbol is on a different layer on the symbols timeline and has a fade out effect at the end. It is basically a tween animation of each graphic symbol that gives the illusion of exploding. You can see it below.



3.2 PADDLE

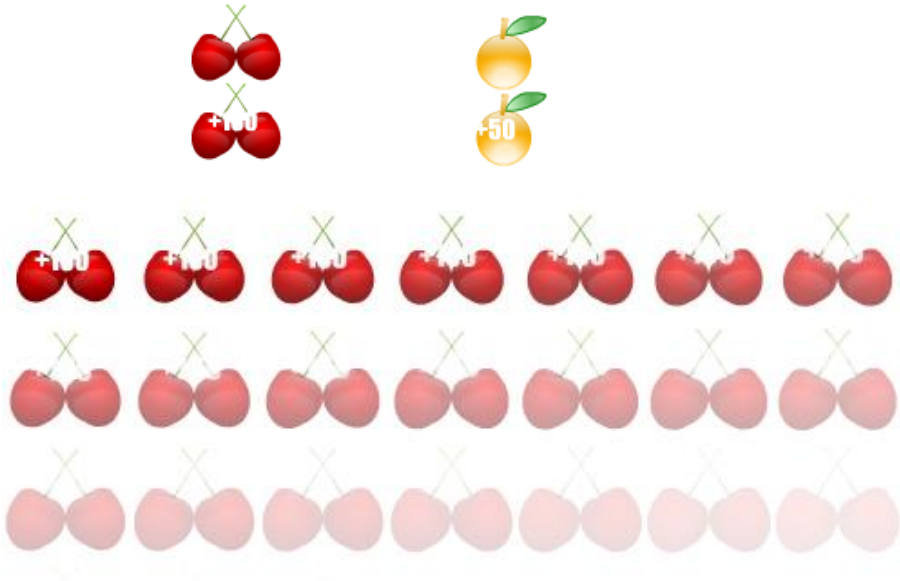
The paddle is a movieclip with no animation attached on it that has several layers used to make it shiny. At [4.4] we will see how the paddle's moving animation works.

3.3 BALL

The ball is also a movieclip with no animation attached on it that has several layers used to make it shiny. At [4.5.1] we will see how the paddle's moving animation works.

3.4 ITEMS

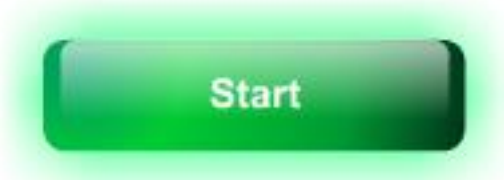
The items are two movieclips with a fade out effect attached. Each item is consists of its fruit and the points as text that are added in the score if the user acquires it. An item is created during game play when a brick is destroyed. Each different brick color produces either a cherry or a mango fruit. At [4.6] we will see how the item's moving animation works.



3.5 OTHER

3.5.1 Start game button

The start button is a movieclip with a text and no animation attached on it that has several layers used to make it shiny. The game is going to start when the button is clicked.



3.5.2 Music toggle button

The play and pause music buttons are movieclips that let the user to control the sounds of the game.

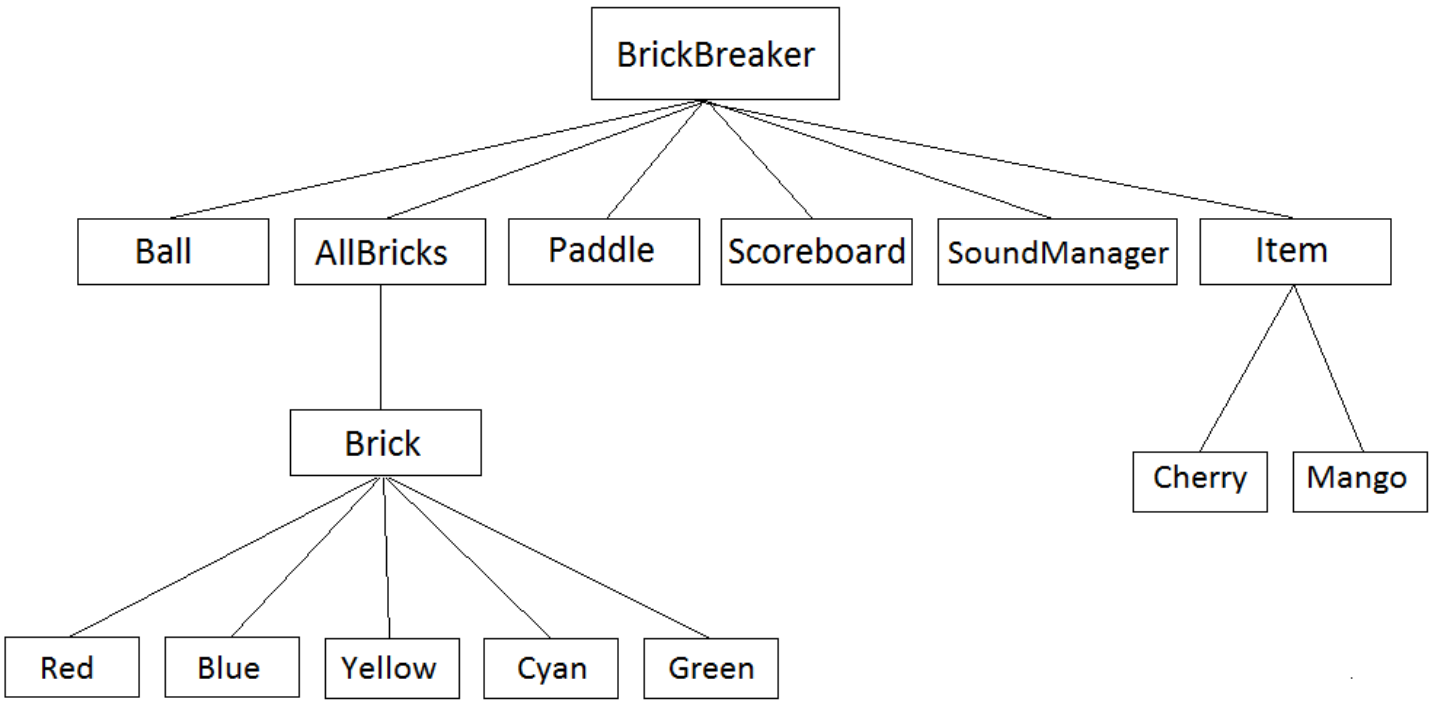
3.5.3 Ending game screens

The screens below are the movieclips that are displayed on the screen when the user either loses or wins. When they are clicked the game is restarted.

4 ACTIONSCRIPT

4.1 A QUICK OVERVIEW

In the following diagram we can see the components that make the game run. Each rectangle represents a class and a connection between two of those means either inheritance or that an object of the first exists in the second. For example the BrickBreaker class has attributes of type Ball, AllBricks, etc. On the other hand a Brick is a superclass of the classes Red, Blue, etc.



4.2 THE BASE CLASS - BRICKBREAKER

The *BrickBreaker* class is the base class of the game and extends the *Sprite* class of the Actionscript 3 library. Every component of the game is an attribute, that exists in this class. In addition to that every class has an attribute of the *BrickBreaker* class to be able to access anything from everywhere. We can see those components below:

allBricks : AllBricks;
Contains an array of *brick* instances, its position is at the top of the Stage and we you could see more about it at [4.3.1].

Paddle : paddle;
Responsible for displaying and moving the paddle with either the mouse or the arrow keys. More about it at [4.4].

Ball : ball;
Includes the implementation for the ball's movement, collision with *bricks* & *paddle*. More at [4.5].

Score : ScoreBoard;
Is responsible for keeping the lives and score values of the user and displaying them. More at [4.8.1].

ballReadyForBouncing : Boolean;
This Boolean variable is responsible for keeping track if the *ball* instance is ready to bounce on the *paddle*. We will see more about it at [4.5].

newGame : Boolean;
A boolean variable that is false when the game has not started and true if it has.

Start : MovieClip;
The start movieclip represents the button that the user needs to click in order to start playing the game. It includes a mouseEvent listener that calls a function which sets the *newGame* Boolean variable to true whenever the object is clicked.

WinorLost : MovieClip;
The variable that is displayed to the user when the game ends. It is either the winning screen if the user wins or the losing screen if the users loses.

soundsHolder : SoundManager;
Contains any functionality needed for managing the sounds of the game, mute and unmute. More about it at [4.7].

ToggleButtonPlay : MovieClip;
This movieclip represents the state of the soundtrack of the game when it is unmuted.

ToggleButtonStop : MovieClip;
This movieclip represents the state of the soundtrack of the game when it is muted.

gameIsPaused : Boolean
The variable that keeps track of whether the game is paused or not. More at [4.8.2]

4.2.1 Initializing the game

The first thing that is created when we run the game is a *BrickBreaker* object. In the constructor of the class the constructors of all the other instances are also called and their positions are set on certain places on the screen. The *gameIsPaused* and *newGame* variables are set to false and the *SoundManager* works as in an unmuted state. As soon as the user clicks the start game button the function *gameBegin* is going to be called and the *start* movieclip will be removed from the screen, the *newGame* variable will be set to true and the function *dispatchevents* is going to be called that is going to add all the necessary event listeners to the stage.

4.2.2 Regarding events

Several events are added to the stage in order for the game to be able to run. Two functions are responsible for those. *dispatchevents* for adding the event listeners to the stage and *removeevents* for removing them. We have two events for controlling the *paddle*, one with the keyboard [4.4.1] and one with the mouse [4.4.2] that call the appropriate functions from the *paddle* class. One event for moving the *ball* [4.5.1]. Two more events are responsible for displaying the score and lives of the player [4.7.1]. Finally, an event for pausing and resuming the game [4.7.2].

4.2.3 Ending the game

When the player loses all their lives or breaks all the bricks on stage the game is finished. The function *GameOver* is responsible for this and displays on the screen the appropriate ending image while playing the appropriate sound. In addition to that the functions are called. *removesymbols* that removes all the compontents that are displayed on the screen and *removeevents* that removes all the listeners that was added on the stage. *GameOver* also adds a mouse listener that calls the function *restart* that reinitializes the game when the image is clicked by the user.

4.3 REGARDING BRICKS

4.3.1 AllBricks

The class AllBricks extends *Sprite* and is a child of the BrickBreaker base class. It is positioned on the point (0,0) of the stage and contains all the bricks that the user need to break in order to win.

bricksNumber:Number;
The current number of bricks on the game.

4.3.2 Initializing the Bricks

The function *attachBricks* extends the *Sprite* class and is used to initialize all the bricks. We run through a loop and we randomly generate a number between 1 and 5 to choose the color(blue, red, yellow, green, cyan) of the next brick. We find the next empty spot and every brick object is added as a child to the AllBricks Sprite.

4.3.3 Brick

Brick extends *MovieClip* and is the super class of all colored bricks. It contains the main functionality for a brick object except from the *DropItem* function which is overridden by its subclasses in order to drop different fruits depending on the color of the brick [4.6].

Hitdetection:Boolean;
It's a Boolean variable that defines if the brick has been hit by the ball during the game and is set to true when the brick explodes.

point:MovieClip;
It's a MovieClip of the points that are added on the score when the bricks break.

4.3.4 The ExplodeMe function

Each brick listens to an event that calls the *ExplodeMe* function when an object enters its frame. When that happens we want to know if the object that entered its frame was the ball. Then we continue with the following. There is a chance of dropping a certain item [4.6] that gives extra score to the user. We create the points MovieClip at the same location that the brick was before it was broken. The direction of the ball is changed vertically [4.5.1]. The function *gotoAndStop* with parameter 2 brings the playhead to a specified frame of the MovieClip and stops it there so that it can continue with playing the breaking animation of the brick. After that we remove the event listener, we increase the score, we decrease the current number of bricks in the game and we play the appropriate sound effect.

4.4 PADDLE

The paddle is the main object that the user controls when the game is not paused and is placed down on the center of the screen. The paddle class extends the *MovieClip* and the user can move it with mouse and keyboard arrows (left and right).

4.4.1 Input through keyboard

The function for moving the paddle with key arrows has as a KeyboardEvent argument to detect the movement. When the paddle is moving on the right side we add a fixed horizontal speed. On the other hand when is moving on the left side we substract a speed 18. On both cases we check if the paddle goes out of the screen and we prevent it.

4.4.2 Input through mouse

The function for moving the paddle with key arrows has as a MouseEvent argument to detect the movement. Here the speed of moving the paddle is controlled by the mouse event. As before we we check if the paddle goes out of the screen and we prevent it.

4.5 BALL

This object is responsible for the ball, its movement, collision with other objects and is practically the core of the game itself. We can see its attributes below.

speedX:Number;
The number that we will add to the position X of the ball on each frame.

speedY:Number;
The number that we will add to the position Y of the ball on each frame.

direction:Number;
The variable that we multiply the above speed variables, when we want to change the direction of the ball, either horizontally or vertically.

lostBall:MovieClip;
This variable contains the text that is displayed on the screen when the ball is lost, or practically goes down of the position Y of the paddle.

4.5.1 Moving the ball

The *moveBall* function is responsible for moving the ball on each frame. This is the function that is called when the ball enters the frames of the stage because of the ENTER_FRAME event listener [4.2.2] , in other words all the time during our game.

If the game is not paused we let the ball move its position by *speedX* and *speedY*.

If the ball reaches the maximum right or left position of the stage, the *speedX* is multiplied with the *direction* variable in order to bounce to the right direction. On the other hand, if the ball reaches the maximum top position of the stage the *speedY* is multiplied with *direction* variable for the same reason. In case of falling down of the paddle, we want to restore the ball on the paddle [4.5.2], refresh the scoreboard [4.8.1] and let the user play again if he has lives left.

The attribute *ballReadyForBouncing* in the *BrickBreaker* class [4.2] is responsible for letting the ball bounce of the paddle when it hits. It is set to false when the ball collides with the paddle, otherwise it is set true.

If a collision with the paddle is detected and the *ballReadyForBouncing* variable's value is true we want to bounce the ball in a different angle depending on the position X of the paddle that it hits. In addition to that, we set the *ballReadyForBouncing* variable's value to false and we play the corresponding sound from the *SoundsManager* [4.7].

The function *changeballAngle* is responsible for the ball's angle during bouncing. It changes the direction of the ball by multiplying the *speedY* variable with the *direction* variable in order to bounce upwards. The *speedX* variable is also multiplied with a certain weight that depends on the position that the paddle was hit. In other words the horizontal speed will be greater if the ball hits at the edges of the paddle and smaller if it hits closer to the middle.

4.5.2 Losing the Ball

When the ball is lost the *balllost* function is going to be called from the *moveBall* function. If the player has not enough lives the LosingScreen is going to pop up and the game will end [4.2.3]. On the other hand, if the player can keep playing the game we display a message that let the user know that he lost the ball with a mouse click event listener attached on it and the event listeners attached to the stage are going to be remove for now.

When the player clicks on it a new ball will bounce of the paddle, the event listeners of the stage are going to be added again and the message on the screen is going to be removed. The X and Y positions of the ball are reset to the positions that the paddle is at the moment and the game will continue.

4.6 ITEMS

There are two items in our game, the mango and the cherry and they sometimes fall off when breaking a brick. The main functionality for any items is contained in the class *Item* that is responsible for making the item to fall down and checks for collision with the paddle. When it collides with the paddle the items are removed from the stage and the player receives the bonus points depending on the type of the item. The *cherry* and *mango* classes are subclasses of the *Item* class and the points that the user receives are defined on the *itemAcquired* function that is overridden by both subclasses. A cherry falls off the blue and the yellow bricks while mango falls from all the others.

An item has a fixed speed value that is added to the position Y of the object in order for it to move downwards when the game is not on a paused state. When the item collides with the paddle or falls down of the paddle it is removed from the stage. If the player successfully acquires it the appropriate sound is going to be played by the *SoundManager*.

4.7 SOUNDS

All the sounds are managed from the *SoundManager* class which extends *Sprite***.** In this class we play all the sounds of the game and also start and pause them. We can see the attributes of the class below. For starting and pausing the music the user can press the “M” key on his keyboard. With a KeyboardEvent we know when the user has pressed the specific key and the music is going to be muted or not. When the game is in a muted state the bottom right Toggle button of the screen displays a play icon. If it is in an unmated state the Toggle button displays a pause icon.

soundtrackIsActive:Boolean;
A Boolean instance that returns if the game sound is active or not.

brickBreaking:Sound;
A sound instance that plays a sound when a brick breaks and the sound of the game is active.

ballBouncing:Sound;
A sound instance that plays a sound when the ball has a collision with the paddle and the sound of the game is active.

powerUp:Sound;
A sound instance that plays a sound when the user acquires an item and the sound of the game is active.

winningSound:Sound;
A sound instance that plays a sound when the user wins and the sound of the game is active.

losingSound:Sound;
A sound instance that plays a when the user loses the game and the sound of the game is active.

4.8 OTHER FEATURES

4.8.1 Scoreboard

The ScoreBoard is managed from *ScoreBoard* class and is placed down and left on the screen. It consists of two Textfields for lives and score. When the user loses the ball the lives are decreased by 1, when the user breaks a brick the score is increased by 10. We also increase the score by 50 and 100 when the user acquires a mango or a cherry respectively.

lifes:uint;
lifes is an uint instance we decrease if the user loses the ball and is initialized with 3.

score:uint;
score is an uint instance we increase every time the user breaks a brick or acquires an item as we mentioned before. The score is initialized with zero.

currentLifesField:TextField;
Is a TextField instance with the current lives of the user and is updated every time the lives are decreased .

currentScoreField:TextField;
Is a TextField instance with the current score of the user and is updated every time the score is increased .

4.8.2 Pausing the game

The user can pause and resume the game pressing the “P” key on his keyboard. With a Boolean instance we know if the game is paused or not. Before letting the movement of the paddle (movePaddleWithMouse / movePaddleWithArrowKeys) ,the ball (moveBall) and the item (moveMe) we check if the Boolean instance is false. If not, those movements are not happening and the game stays on at paused state until the key “P” is pressed again.