BLOCK BREAKER

Documention& Manual

❖ Manual

- How To Play

❖The Code

- The Platform
- The Ball & Collision System
- Blocks
- World initialization
- Game Loop

❖Classes

Manual

In order to play the Game no setup of any kind is needed

Just Unpack the zip file and run the exe

There will be a splash screen that you can pass by clicking anywhere on the screen. Afterward the main menu will appear which you can choose to start or quit the game from.

After clicking on the start button the game will begin. The goal if the game is to break as many as the bricks possible without letting the ball fall.

The Code.

The Game consist of numerous objects. Each object has a spirit which moves or changes according to the events happening in the game.

You can see the class diagram at the end of this document.

The Platform (Paddle)

We start by explaining the system of the platform. platform is a device player can move to avoid the loosing of ball.

Platform has an smooth movement system .meaning it movements is smooth and its velocity changes based on the time the player holds the arrow buttons .

Every time an arrow key is pressed the voidpadel::timerStart () of the padel class starts and when the key is realesed the stop() method is called .

The value of the timer is used in voidpadel::update as its shown here :

```
//timer indicates how long we have hold the left/right buttons
voidpadel::goLeft()
{
	direction = 1 ;
	Xvelocity = timer.getElapsedTime().asSeconds()+0.5 ;
	if(getX() - 0.5*Xvelocity>= 0)
		setPositon(getX()-0.5*Xvelocity,460);
}
```

In order to detect the released or pressed status of the arrow keys both real time key status and event driven key status have been used.

Although the event driven part is exclusively used only to detect releasing or pressing the keys and therefore calling the timerStart() or stop() methods .

```
while(GameWindow.pollEvent(GameWindowEvent))
	{	if(GameWindowEvent.type==sf::Event::EventType::KeyPressed&&
		(GameWindowEvent.key.code == sf::Keyboard::Left|| GameWindowEvent.key.code
		== sf::Keyboard::Right))
			{
				PlayerPadel.timerStart();
			}
		if(GameWindowEvent.type==sf::Event::EventType::KeyReleased&&
			(GameWindowEvent.key.code == sf::Keyboard::Left||
			GameWindowEvent.key.code == sf::Keyboard::Right))
			{
				PlayerPadel.stop();
			}}
```

The Ball & Collision System

The ball is based on a angular movement system .meaning instead of moving by coordinates , it moves a certain amount in the direction of the angle given to it .

To do so we need to use the mathematical sin() and cos() of the cmath library on the angle given to the ball as you can see below.

```
// make the angle stay between -360 & +360
while(angle>360)
	angle = angle -360 ;
while(angle<-360)
	angle = angle +360 ;

//converting degree to gradian
floatdegree = (angle*3.141)/180 ;

//calculating the new positon
posx = posx + 0.2*std::sin(degree);
posy = posy + 0.2*std::cos(degree);
```

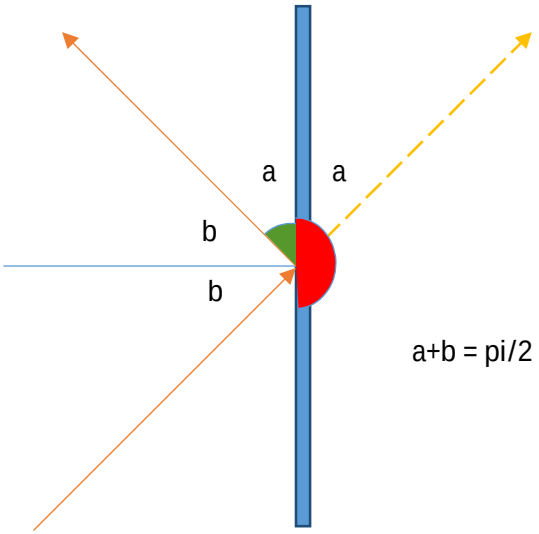
This system makes the ball movement a lot easier. so in order to change the ball direction all we need to do is to change the angle given to the ball in the next frames .

Ball direction can change with collision to the upper and side wall, the platform, and the blocks.

There are mainly two kind of collisions .vertical and horizontal .

Each one is described dividedly.

Vertical collisons (side walls)



As its shown in the following diagram in the vertical angle follows the simple formula

Angle = (-angle)

As shown in the code below :

```
//collision with left & right walls
if (posx> 500 || posx< 0)
angle = - angle ;
```

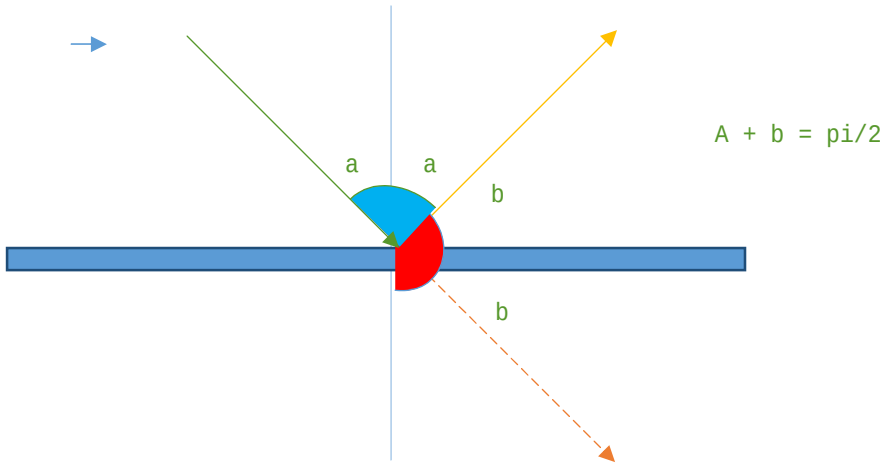
Horizontal Collisions

the horizontal collision are a little trickier since every angle is shifted by pi/2 . so the new formula for the horizontal collisons is : angle = angle – 2*(angle -90)

as shown in the codes below for the collisions with the paddle or the upper walls.

```
//collision with up wall
	if(posy<0 )
		angle = angle - 2*(angle-90) ;
//collision with padle
	if(( posy>PlayerPadel.getY()-10
		&& (posx>= PlayerPadel.getX()
		&&posx<PlayerPadel.getX()+100)))

	{
		angle = angle - 2*(angle-90) +
	PlayerPadel.direction*PlayerPadel.getXVelocity()*30 ;
	}
```



As it can be seen above the collision with the paddle has something extra :
Based on the velocity of the paddle which was discussed in the perevious chapter, an amount based on degree is added to the ball's current angle .so that the velocity can affect the ball direction like in the real world experiment .

```
angle = angle - 2*(angle-90) + PlayerPadel.direction*PlayerPadel.getXVelocity()*30 ;
```

Blocks

Blocks have a simple structure
They just store some coordinates and whenever the ball cross that coordinate AND the block is not already broken which is acknowledged by checking the isExist flag, the ball bounces back using the same horizontal collision .
After that they also add 1 to the brokenBlocksCount variable of the ball object .this will later be used to calculate the score .
The following code in the Update() method of the ball checks every blocks in every frame to see if the collision happens :

```
for (inti = 0 ; i< 5 ; i++)
	{
		for (intj = 0 ; j< 5 ; j++ )
		{
		if(posy<blocks[i][j].getY()+20
			&&posx>blocks[i][j].getX()
			&&posx<blocks[i][j].getX()+100
			&&blocks[i][j].doesExist
			)
			{
				angle = angle - 2*(angle-90);
				blocks[i][j].doesExist = false;
				brokenBlocksCount +=1 ;
			}
		}
	}
```

World initialization

In this chapter we explore the constructor of the world class .

When an instance of world class is created , the first thing that happens is creation of a windows form which the game happens in .

Then an instance of the splashMenu&mainMenu classes are created and drawn to the windows form .

this is done by the following line :

```
// creating the game window
	GameWindow.create(sf::VideoMode(500,500),"Block Breaker v0.2a");


// showing the splash screen
	GameSplashScreen = newSplashScreen ;
	GameSplashScreen->show("welcome.png",GameWindow);
	deleteGameSplashScreen ;

//showing the main menu
	MenuScreen = newmenu ;
	MenuScreen->show(GameWindow);
	deleteMenuScreen ;
```

notice that since the world object never gets destructed , we used pointers to manually deallocate things we no longer will need such as these menus .

after that , first the score is set to 0 by the line :

```
//setting up the score ( to 0      )
showScore(0);
```

then a sf::Text object is created to represent the score value later .

```
sf::FontscoreFont ;
scoreFont.loadFromFile("dlxfont.ttf");
score.setFont(scoreFont);
score.setPosition(20,450);
```

this representation is done by the showScore method of the World class :

```
voidworld::showScore(intscr)
{
	scoreString = std::to_string(scr) ;
	score.setString(scoreString);
	GameWindow.draw(score);
}
```

And lastly , the game blocks are created using a for loops , then the game loop begins .

```
//setting up the blocks

sf::TextureBlockTexture;
BlockTexture.loadFromFile("block.png");
for (inti = 0 ; i< 5 ; i++)
	{
		for (intj = 0 ; j< 5 ; j++ )
		{
			blocks[i][j].GameObjectSprite.setTexture(BlockTexture);
			blocks[i][j].setPositon(i*100,j*25);

		}
	}
```

The Game Loop

The game loop consist of two part :

I. Updating every game object

This is done by lines :

```
PlayerPadel.update(GameWindow);
GameBall.update(GameWindow,PlayerPadel,blocks);
```

The update() method for the GameBall was explained earlier . it moves the Ball in a specific angle by a certain amount every time the gameloop runs or in other words every frame.

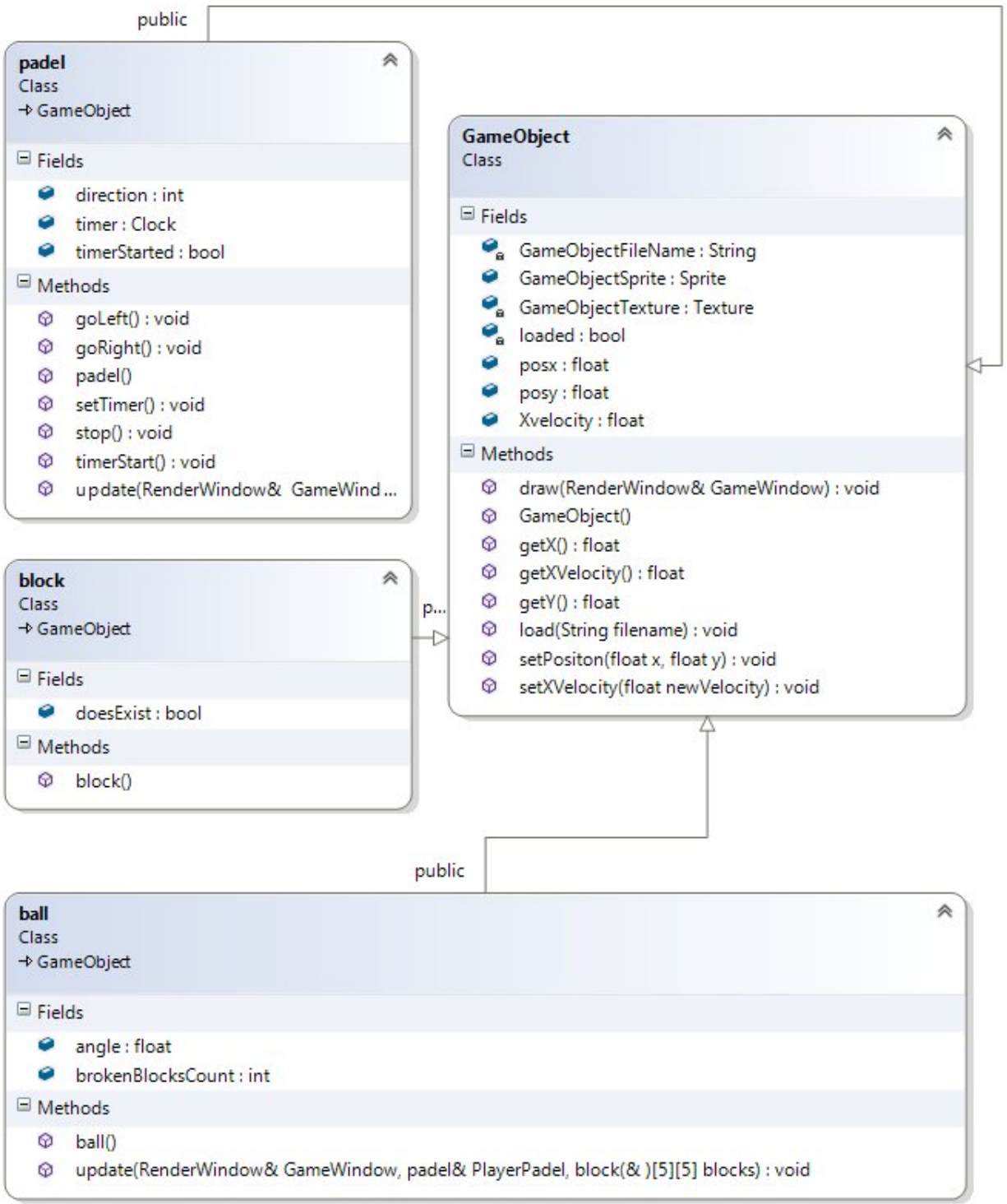
The update() method for the PlayerPadel does something similar. It moves the platform a certain amount based on platform velocity .also it updates the platform velocity every frame .

II. Drawing every game object to the game window

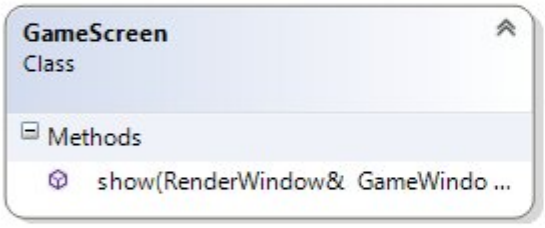
This is done by the following lines :

```
GameWindow.clear(sf::Color(100,30,240));
PlayerPadel.draw(GameWindow);
GameBall.draw(GameWindow);
drawBlocks();
showScore(GameBall.brokenBlocksCount*10);
sf::sleep(sf::microseconds(300));
GameWindow.display();
```

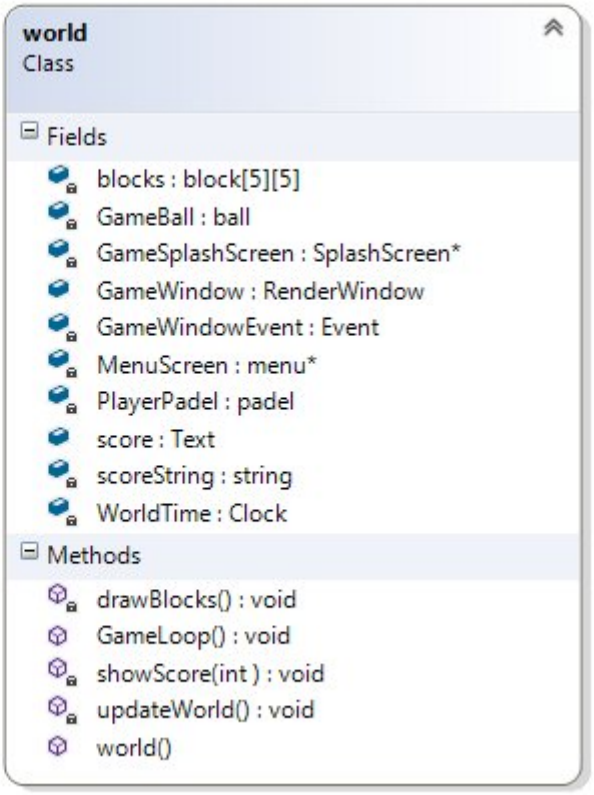
Classes


public
padel
Class
GameObject
Fields
direction : int
timer : Clock
timerStarted : bool
Methods
goLeft() : void
goRight() : void
padel()
setTimer() : void
stop() : void
timerStart() : void
update(RenderWindow& GameWind ...
GameObject
Class
Fields
GameObjectFileName : String
GameObjectSprite : Sprite
GameObjectTexture : Texture
loaded : bool
posx : float
posy : float
Xvelocity : float
Methods
draw(RenderWindow& GameWindow) : void
GameObject()
getX() : float
getXVelocity() : float
getY() : float
load(String filename) : void
setPositon(float x, float y) : void
setXVelocity(float newVelocity) : void
block
Class
GameObject
p...
Fields
doesExist : bool
Methods
block()
public
ball
Class
GameObject
Fields
angle : float
brokenBlocksCount : int
Methods
ball()
update(RenderWindow& GameWindow, padel& PlayerPadel, block(&)[5][5] blocks) : void


menu
Class
Methods
show(RenderWindow& GameWindo ...


GameScreen
Class
Methods
show(RenderWindow& GameWindo ...


SplashScreen
Class
Methods
show(String fileName, RenderWindo...


world
Class
Fields
blocks : block[5][5]
GameBall : ball
GameSplashScreen : SplashScreen*
GameWindow : RenderWindow
GameWindowEvent : Event
MenuScreen : menu*
PlayerPadel : padel
score : Text
scoreString : string
WorldTime : Clock
Methods
drawBlocks() : void
GameLoop() : void
showScore(int) : void
updateWorld() : void
world()