Odd Semester (2021)

Assignment Cover Letter

(Individual Work)

Student Information:	**Surname**	**Given Name(s)**	**Student ID Number**
1.	**Redzwan**	**Alfi**	**2101693574**

Course Code	**: COMP6502**	**Course Name**	**: Introduction to Programming**
Class	**: L1BC**	**Name of Lecturer(s)**	**:** 1. Jude Martinez 2. Minaldi Loeis
Major	**: CS**		

Title of Assignment (if any): Hardcore brick Breaker

Type of Assignment: Final Project

Submission Pattern

Due Date	**: 7-11-2017**	**Submission Date**	**: 7-11-2017**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

Binus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to Binus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

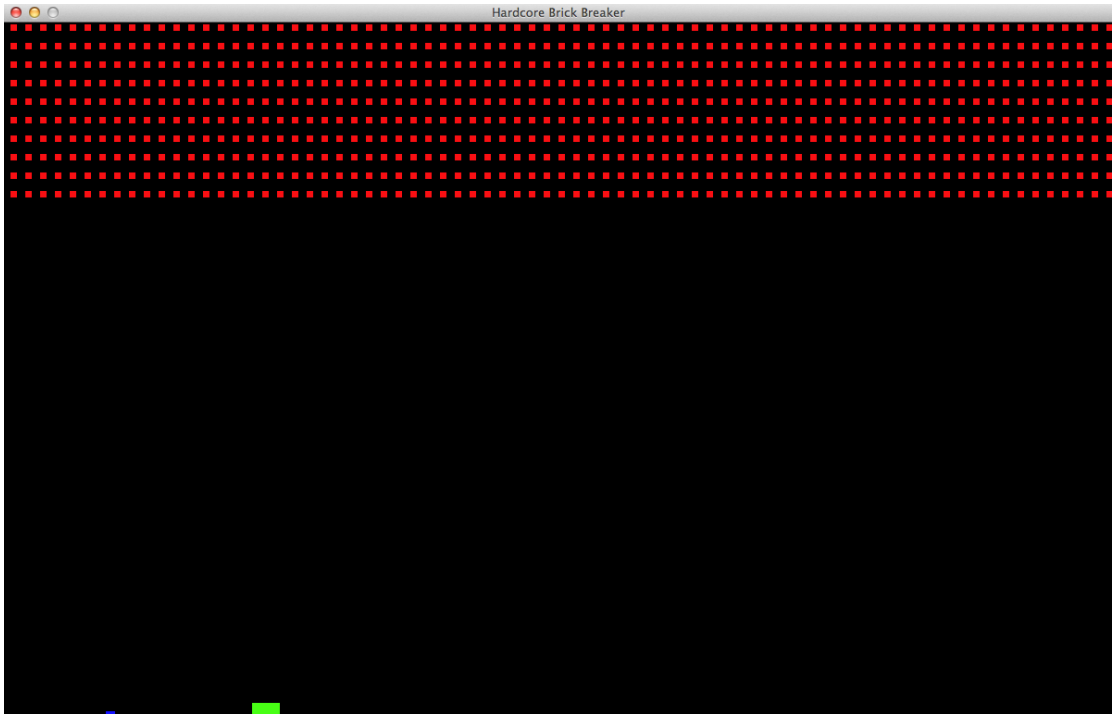
Signature of Student: (Name of Student)

1 Alfi Mohamed Redzwan

“Hardcore Brick Breaker”

Name: Alfi

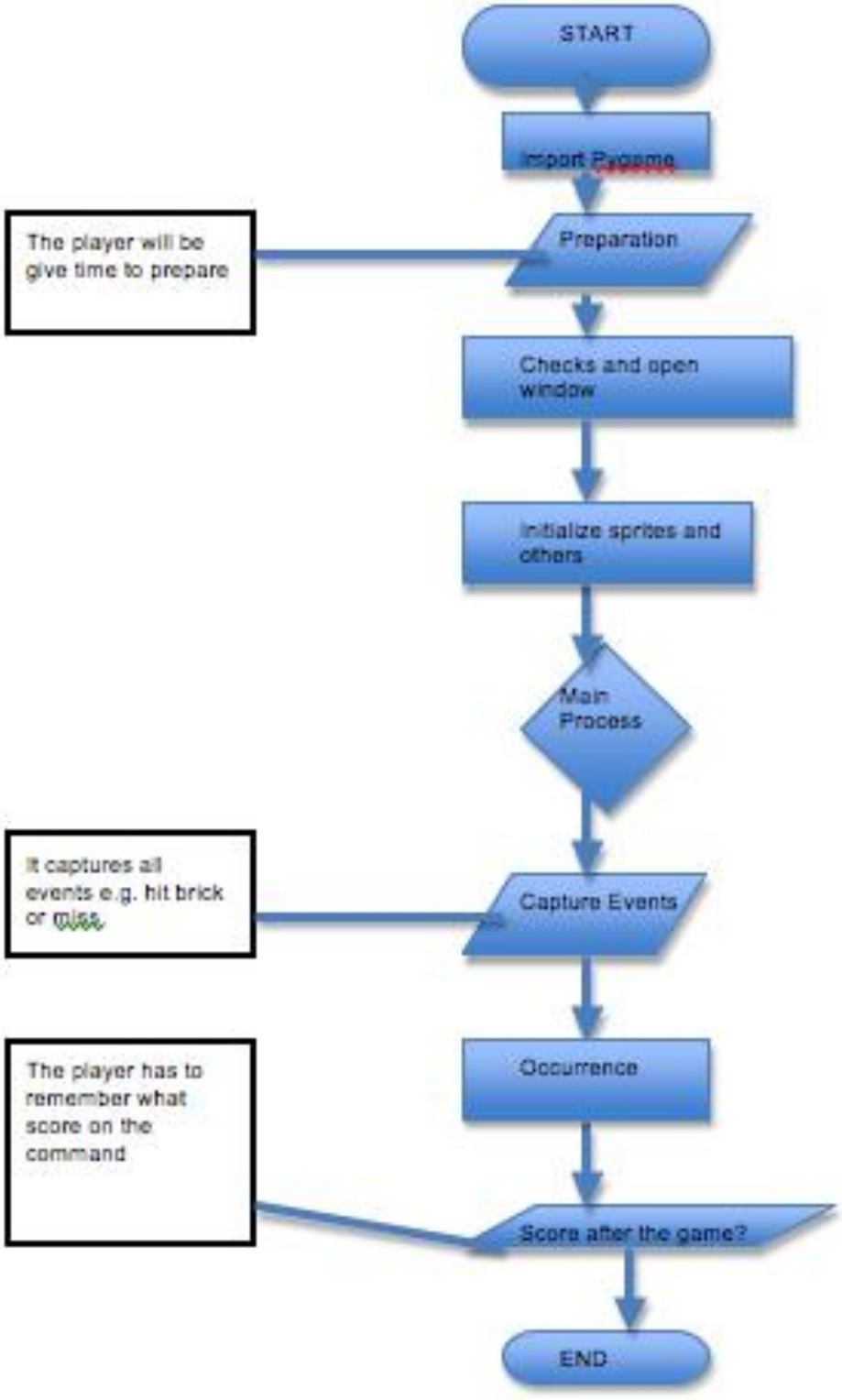
ID: 2101693574



I. Description

This program is to help people learn how to make a good programming language using pygame as a python. It can help people make sprites, screen, main menu by using classes, functions and also inheritance. In addition, it can learn when to print and use input in specific events.

II.a. Design(Flowchart of Pygame)



IIb. Flowchart explanation

START: Start the program of the game. It reads all the codes that have written on Python.

Preparation: This code gives you time to prepare yourself if you want to put other activities away before preparing the game.

Initialize sprites and others: It initializes the classes including the sprites and the function of how it does.

Main Process: Loops and updates the game every action it does. It continues until it says game over. First it shows the main menu which means the game is about to start and then is starts the main process

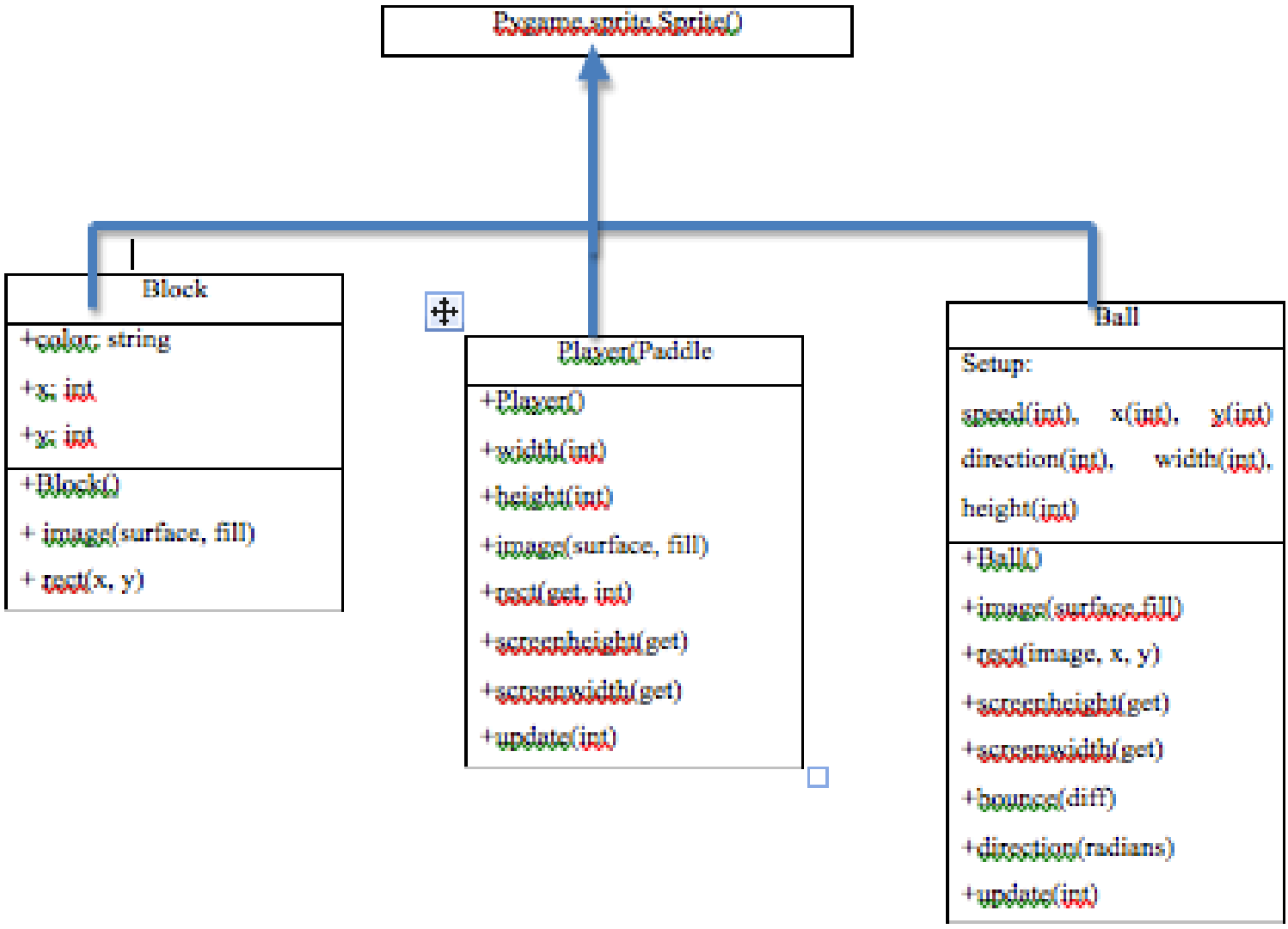
Capture Events: The console command will print every action of each thing it does e.g. hitting a brick

Occurrence: How often does it occur? Is it rarely? Is it often?

Score after the game: Provides how many scores got if it’s game over or the level is completed

END: Ends the main process of the game

UML Diagram


Pygame.sprite.Sprite()
Block
+color: string
+x: int
+y: int
+Block()
+ image(surface, fill)
+ rect(x, y)
Player(Paddle
+Player()
+width(int)
+height(int)
+image(surface, fill)
+rect(get, int)
+screenheight(get)
+screenwidth(get)
+update(int)
Ball
Setup:
speed(int), x(int), y(int) direction(int), width(int), height(int)
+Ball()
+image(surface,fill)
+rect(image, x, y)
+screenheight(get)
+screenwidth(get)
+bounce(diff)
+direction(radians)
+update(int)

III. Steps

a. Introduction

The first time I made the final project is using pygame. The main source code is to make the sprites, making the main menu and sprites. It's not just the main source code, check also the main events that are running whether it's updating or not.

I've decided to make Brick Breaker as part of my final project. I've put the class as an inheritance for every sprites which is the part of my plan. First, I put the bricks as part of the main completion for the game. Second, I've put the paddle as the main control for the player to keep the game running. Third, the ball is the part whether the ball goes off the screen to check if it's game over or not. Lastly, I've decided to put the music that fits the difficulty of the setting. All of the sprites are classified as a subclass. In this case, I've made the game as hardcore and I've tried to put intense music that fits the difficulty. The objective is to break all the bricks.

b. Explanation of each class

Main Menu Screen

Before making the sprite, I typed the following code to make the screen.

```
import pygame

pygame.init()

# Creates the screen according to the size
screen = pygame.display.set_mode([1200, 750])

# Set's the caption
pygame.display.set_caption('Hardcore Brick Breaker')

# Create a surface
background = pygame.Surface(screen.get_size())

# Create sprites and screen
allsprites = pygame.sprite.Group()

#Draw sprites and screen
    allsprites.draw(screen)
```

```
    # Flip the screen and show what's drawn
    pygame.display.flip()

pygame.quit()
```

The Import code functions import system from the packages or files. The pygame.display.set_caption() sets the title of the game on top of the screen. The pygame.display.set.mode() uses to measure the screen by typing the given numbers. The clock.tick() shows the count of the fps(frame rates per second) that was playing on the screen. The pygame.Surface() represents images and screen. Pygame.quit() ends the process of the game. pygame.sprite.Group() and pygame.draw() will be explained later.

How does the program know when to close the game? Make sure to put the main program loop:

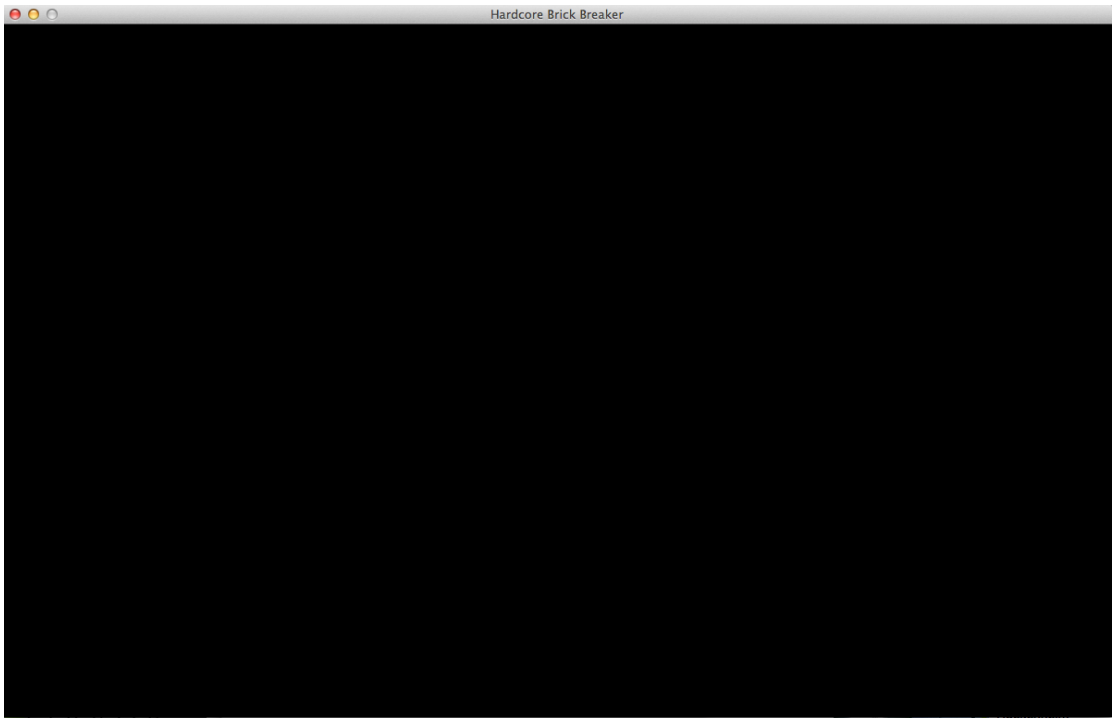
```
Import pygame
-(snip)-
clock = pygame.time.Clock()
# Exit the program?
exit_program = False

# Main program loop
while not exit_program:

    # Fps setup
    clock.tick(30)

    # Fills the screen
    screen.fill(black)

    # Process
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit_program = True
-(snip)-
```

The main variable I put is exit_program. First I put exit_program=False is known that the program is still running if the user exits the game while running put True. The pygame.event.get() captures the events running in the game. The QUIT statement is that the program stops if the player exits the game.

Don't forget to put the colors for the screen by using screen.fill(color name) which fills the screen. There are lots of colors given in pygame. For instance here is an example:

```
black = (0, 0, 0)
white = (255, 255, 255)
blue = (0, 0, 255)
```

Make sure the number of the color must be the same to match the type otherwise it will not show. Put pygame.display.flip() to update the display on the screen. And don't forgot to put pygame.init() before the while loop to make sure that it's working. Because that code initializes all imported modules. The while loop I put is to know when to exit the program if the player is done playing.



Making Sprites

After making the screen, I've made the following sprites, which is ball, paddle and the block, the basic sprites for brick breaker. For instance I've used them as a subclass and I've put the command pygame.sprite.Sprite as the main class for drawing the sprite.

Block

Make the Block class by setting the image by using rect x and y. Rect draws the sprite, if not, then it will not show. To set up the row and column for the block, use for in range loop. Before doing that I've made a variable name blockcount to put how many blocks given. Always end the class with pygame.init(). Self.image pops up the block.
Import pygam

Making Sprites

After making the screen, I've made the following sprites, which is ball, paddle and the block, the basic sprites for brick breaker. For instance I've used them as a subclass and I've put the command pygame.sprite.Sprite as the main class for drawing the sprite.

Block

Make the Block class by setting the image by using rect x and y. Rect draws the sprite, if not, then it will not show. To set up the row and column for the block, use for in range loop. Before doing that I've made a variable name blockcount to put how many blocks given. Always end the class with pygame.init(). Self.image pops up the block.

```
Import pygame
-(snip)-
class Block(pygame.sprite.Sprite):
def __init__(self, color, x, y):
super().__init__()
self.image = pygame.Surface()
self.image.fill(color)
self.rect = self.image.get_rect()
self.rect.x = x
self.rect.y = y
```

To determine the height and width, type the command:

```
import pygame
block_width = 600
block_height = 15
-(snip)-
```

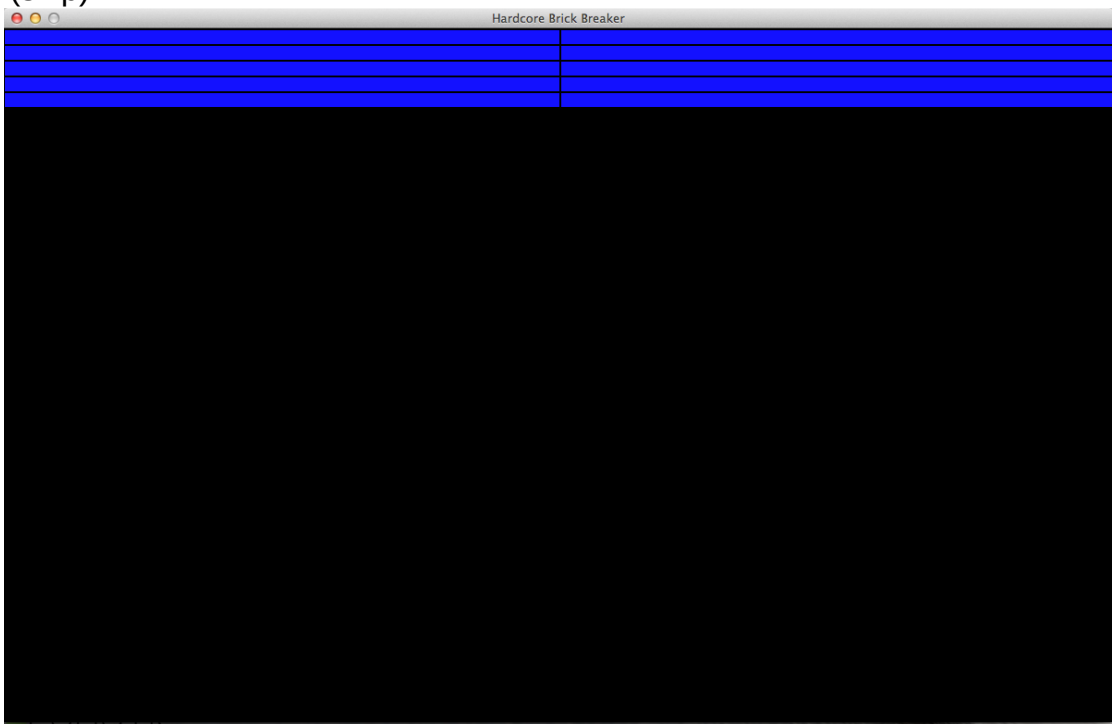
It can be any size as long as it's hardcore.

How does it know when to put the position of the block(y)? Put top to position the block. The blockcount variable determines how many blocks. To make the determine how many rows and I've put for in range to make sure that it counts the rows and columns as mentioned before. The .add command is to draw sprites. For row in range is to add the number of rows while columns is to add column. The variable allsprites will be explained later.

```
Import pygame
-(snip)-

# Position of block (y)
top = 2

# How many blocks to put
blockcount = 10

# --- Create blocks

# Sets the rows and columns
```

```
for row in range(5):
    for column in range(0, blockcount):
        # Create a block (color,x,y)
        block = Block(blue, column * (block_width + 2) + 1, top)
        blocks.add(block)
        allsprites.add(block)
    # Move the top of the next row down
    top += block_height + 2
```

-(snip)-



Paddle

This is the main subclass for the Player to control the process of the game. It uses the same class as the Block. For instance type Player as the paddle to know that if the player is controlling the paddle. Setup the width and height of the paddle that it's comfortable for game play. This class code has the same code as the Block. In addition, the pygame.mouse.get_pos() captures the position of the mouse in the game and for instance as the main control of the paddle.

```
Import pygame
-(snip)-
class Player(pygame.sprite.Sprite):
def __init__(self):
super().__init__()
self.image = pygame.Surface([self.width, self.height])
self.image.fill((yellow))
```

```
# Make our top-left corner the passed-in location.
self.rect = self.image.get_rect()
self.screenheight = pygame.display.get_surface().get_height()
self.screenwidth = pygame.display.get_surface().get_width()
self.rect.x = 0
self.rect.y = self.screenheight-self.height
```

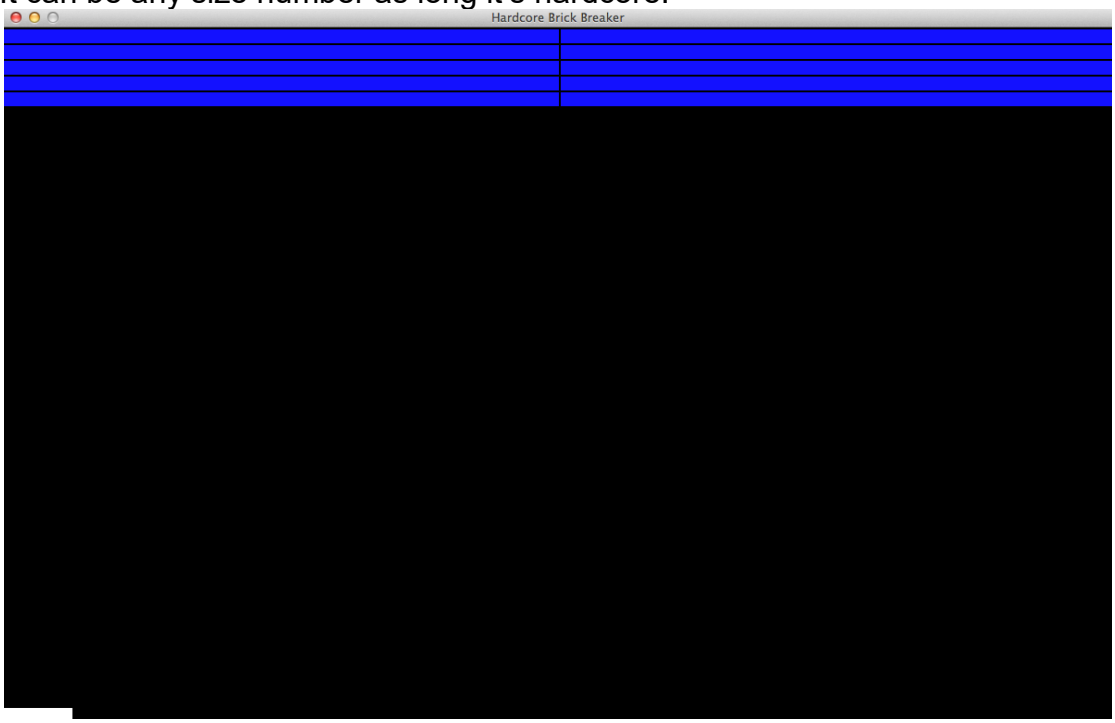
How does it know if the paddle goes off the screen or not? Surely not. Because in real brick breaker make sure that the paddle does not goes off the screen. Put this command. The update means that the sprites always whether it goes left or right according to the player's control this event was also included in Ball.

```
def update(self):
pos = pygame.mouse.get_pos()
# Set the left side of the player bar to the mouse position
self.rect.x = pos[0]
# Make sure to not push the player paddle
# off the right side of the screen
if self.rect.x > self.screenwidth - self.width:
self.rect.x = self.screenwidth - self.width
```

To determine the height and the width type:

```
self.width = 75
self.height = 15
```

It can be any size number as long it's hardcore.



Ball

Next I've put the ball to make the paddle to bounce the ball to complete the level. Usually the class uses the same thing as Block. If the ball touches the wall or a brick, put the function bounce to know where it bounces next. The code class is the same as before but this time I've added self direction for the ball to detect if it's bounces from the paddle or the brick which be explained later.

```
class Ball(pygame.sprite.Sprite):
def __init__(self):
super().__init__()
self.image = pygame.Surface()
self.image.fill(green)
self.rect = self.image.get_rect()
def bounce(self, diff):
self.direction = (180 - self.direction) % 360
self.direction -= diff
```

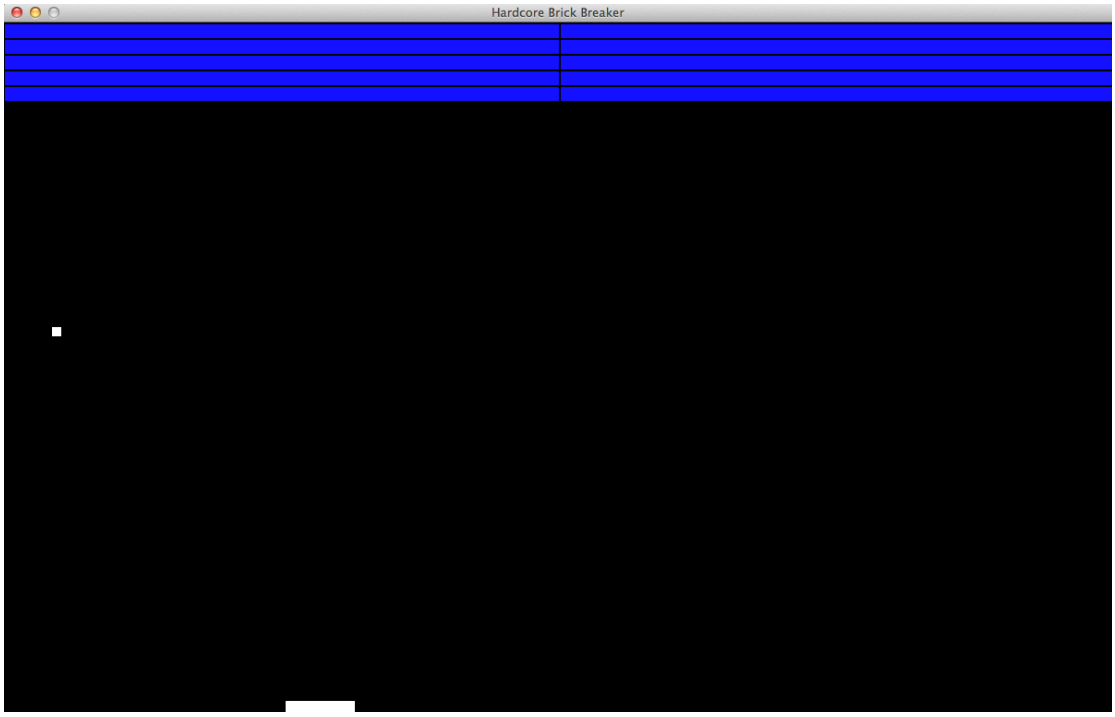
The size of the ball is also the same as the paddle.

```
width = 10
height = 10
```

To know how the ball is bouncing or not is that I've put self.bounce by using update similar to Paddle. But first, import math so that the game can calculate which direction the ball is going. The sin and cos value returns the radiant's of sin and cosine respectively according to the direction of the ball. The self.speed code determines the speed of the ball times the value of the sin cos so that the ball can go diagonally, horizontally or vertically.

```
import pygame
import math
def update(self):
direction_radians = math.radians(self.direction)
self.x += self.speed * math.sin(direction_radians)
self.y -= self.speed * math.cos(direction_radians)
self.rect.x = self.x
self.rect.y = self.y
if self.y <= 0:
self.bounce(0)
self.y = 1
if self.x <= 0:
self.direction = (360 - self.direction) % 360
self.x = 1
if self.x > self.screenwidth - self.width:
self.direction = (360 - self.direction) % 360
self.x = self.screenwidth - self.width - 1
if self.y > 800:
return True
else:
return False
```

Put return True or False if the ball is bouncing and going on or off the screen



Showing all sprites

Finally, I've used command to make all the sprites appear. The .add command is used to add the sprite into the program. Pygame.sprite.group is to make each of the sprites appear from each variable. Allsprites shows all the sprites by using pygame.draw.

```
# Create sprite lists
blocks = pygame.sprite.Group()
balls = pygame.sprite.Group()
allsprites = pygame.sprite.Group()
# Create the player paddle object
player = Player()
allsprites.add(player)
# Create the ball
ball = Ball()
allsprites.add(ball)
balls.add(ball)
```

Game play

The first things it too make the mouse cursor invisible by typing: pygame.mouse.set_visible(0)

How does it know if the ball hits the brick? I've typed:

```
# Collision between ball, brick and paddle
if pygame.sprite.spritecollide(player, balls, False):
# The 'diff' lets you try to bounce the ball left or right
```

```
# depending where on the paddle you hit it
diff = (player.rect.x + player.width/2) - (ball.rect.x+ball.width/2)
# Set the ball's y position in case
# Hits the ball on the edge of the paddle
ball.rect.y = screen.get_height() - player.rect.height - ball.rect.height - 1
ball.bounce(diff)
# Check for collisions between the ball and the blocks
deadblocks = pygame.sprite.spritecollide(ball, blocks, True)
# If it hits, bounce the ball
```

Pygame.sprite.spritecollide detects if the sprites is hitting the brick or paddle or not. If it's not I've put False if it is, put True. Diff code calculates the distance between the paddle and the ball on the screen from player.rect, player.width and ball.width,ball.rect.
To make sure how the blocks will break I put the command:

```
if len(deadblocks) > 0:
        sound.play()
        ball.bounce(0)
        print("hit")
```

The len deadblocks determines the length of the brick. Make sure to put greater than zero to make the game continue running.

```
import pygame
-(snip)-
if len(deadblocks) > 0:
ball.bounce(0)
```

pygame.sprite.collide shows that the spirte makes contact with each other.

This command shows if the ball hits the brick, it will bounce according to the direction of the brick.

Game over

How does it know when will the game ended or the level was completed?. Put the variable game_over. To keep the game playing put the false statement on game over. Also, put if statements on the paddle and the brick class. Also put the text "Game over" by typing text as a variable and textpos. Textpos is to determine which position is gonna set. Don't forget to put pygame.quit() for the game to close. Also put the text before the game_over code. Font.render used to appear the text in the event which was the type of font and the color. But first I've put pygame.font.Font which puts the word given and the size. Remember always update the game including the paddle and the ball by using ball.update() and player.update().

```
Import pygame –(snip)-
font = pygame.font.Font(None, 36)
Import pygame
-(snip)-
#How does it know when it's game over
```

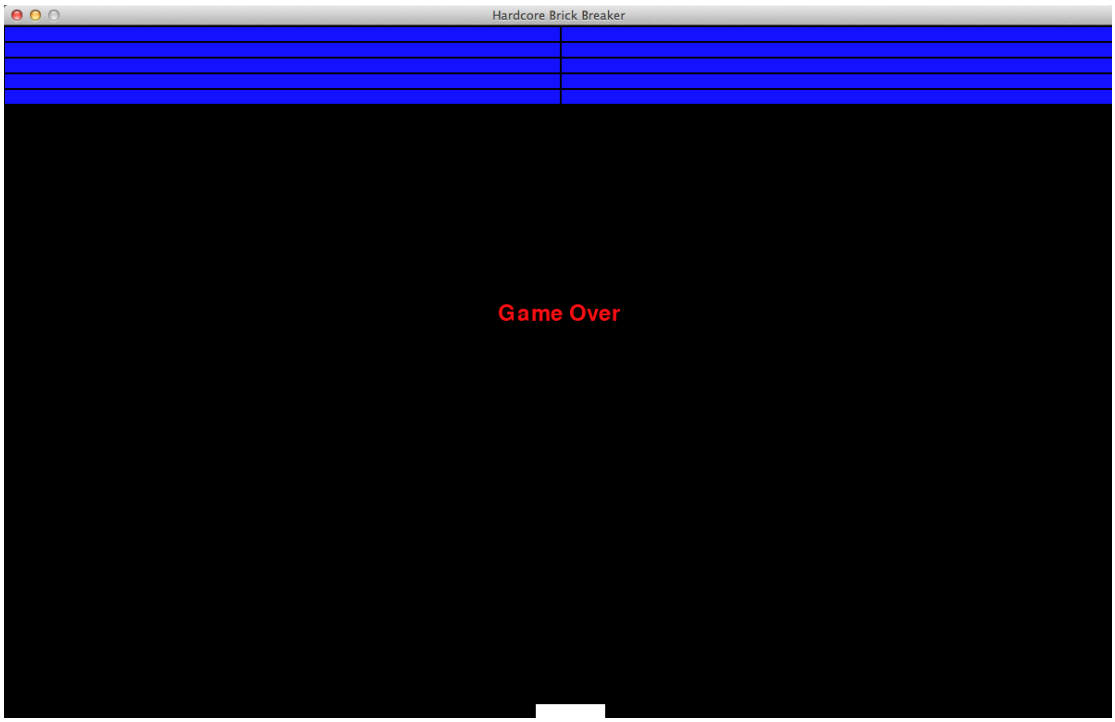
```
game_over = False
if not game_over:
# Update the player and ball positions
player.update()
game_over = ball.update()
# If done, print game over
if game_over:
text = font.render("Game Over", True, white)
textpos = text.get_rect(centerx=background.get_width()/2)
textpos.top = 300
screen.blit(text, textpos)
```

Put this code to make sure that the level is completed:

```
    if len(blocks) == 0:
        game_over = True
        print("Level completed")
```

The code if not statement for game over determines that the game is not over because the ball does not goes off the screen. The screen.blit() displays the text given from the previous code. Also I've decided to put the text to show that the game ended. And the position can be either middle top or middle. For instance I put top.

If the brick were cleared, noted that is says game over on the screen if the player completes the level, I put print as level completed to make sure that the level is completed.



Music and Sounds

To make the game more comfortable, I've decided to put music and sounds just like in the real brick breaker. For instance use:

```
pygame.mixer.init()
pygame.mixer.music.load(file)
pygame.mixer.music.play(-1)
sound = pygame.mixer.Sound("break3.wav")
effect = pygame.mixer.Sound("Paddle ball hit 1.wav")
```

In the given code the pygame.mixer.music.load(file) loads the music given from the library. The pygame.mixer.music.play shows that the music is playing in the game. Make sure to put .load before playing, other wise it will not play. The -1 statement in play repeats the music Pygame.mixer.sound plays the sound and make sure sure to put sound.play() on the respective commands. The sound is acts the same as music but make sure to put it at respective code.

IV. Problems that I've had when making the program

During that day before next week that is the due date, I've experienced lots of problem of making the program. The first problem I've experience is putting the music. And I've learned that some of the files did not work for each of them. I have no problem of putting music. The second and main problem is that I have no idea how to make a button although I've already made the main menu of the program. Although I have an idea inspired by other games rather than brick-breaker. My idea to make an empty main menu to give them a second chance prepare if they're busy. And mote that I put sys and pygame.locals as a comment because I was testing for putting images but then it was too difficult and sys for exit the whole program. In the end, I've decidec not to put on them for the sys and pygame.locals.

V. Full Source Code

This is my actual code of the Brick Breaker. Because in the previous steps, that was just a guide of making the brick breaker and then it can be edited later for the Brick, Paddle and Ball class to make it hardcore and this time there's no HUD.

```
#From Paul Vincent Harvent
#Note the classes that I put are in one file.
import math
import pygame
import time
```

```
#import sys
#from pygame.locals import *
# Colors added and block setup
black = (0, 0, 0)
white = (255, 255, 255)
blue = (0, 0, 255)
red = (255, 0, 0)
green = (0, 255, 0)
yellow = (255, 255, 0)
block_width = 7
block_height = 7
#Prepare for the game to start
print("Hardcore Brick Breaker")
time.sleep(3)
print("Ready")
time.sleep(5)
print("This game is set to the hardest difficulty. Do your best \n Pre-game setting: Exit to Start \n
Setting: HUD DISABLED \n Small Paddle \n High Speed Ball")

#Creates the Block Sprite
print("Game started")
class Block(pygame.sprite.Sprite):
    def __init__(self, color, x, y):
        super().__init__()
        self.image = pygame.Surface([block_width, block_height])
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.x = x
        self.rect.y = y
#Creates the Paddle Sprite
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        #Block setup for width and height
        self.width = 30
        self.height = 15
        self.image = pygame.Surface([self.width, self.height])
        self.image.fill((green))

        # Make our top-left corner the passed-in location.
        self.rect = self.image.get_rect()
        self.screenheight = pygame.display.get_surface().get_height()
        self.screenwidth = pygame.display.get_surface().get_width()

        self.rect.x = 0
        self.rect.y = self.screenheight-self.height

    def update(self):
        pos = pygame.mouse.get_pos()
        # Set the left side of the player bar to the mouse position
        self.rect.x = pos[0]
        # Make it doesn't push the player paddle
        # off the right side of the screen
        if self.rect.x > self.screenwidth - self.width:
            self.rect.x = self.screenwidth - self.width

#Determines the classification of the ball
class Ball(pygame.sprite.Sprite):
#Ball setup
    speed = 15.0
    x = 200.0
    y = 500.0
    direction = 200
    width = 10
    height = 10
```

```
    def __init__(self):
        super().__init__()
        self.image = pygame.Surface([self.width, self.height])
        self.image.fill(blue)
        self.rect = self.image.get_rect()
        self.screenheight = pygame.display.get_surface().get_height()
        self.screenwidth = pygame.display.get_surface().get_width()


    def bounce(self, diff):
        self.direction = (180 - self.direction) % 360
        self.direction -= diff
#Updates the game
    def update(self):
        direction_radians = math.radians(self.direction)
        self.x += self.speed * math.sin(direction_radians)
        self.y -= self.speed * math.cos(direction_radians)
        self.rect.x = self.x
        self.rect.y = self.y


        if self.y <= 0:
            self.bounce(0)
            self.y = 1


        if self.x <= 0:
            self.direction = (360 - self.direction) % 360
            self.x = 1


        if self.x > self.screenwidth - self.width:
            self.direction = (360 - self.direction) % 360
            self.x = self.screenwidth - self.width - 1


        if self.y > 800:
            return True
        else:
            return False



#Put Music that matches the difficulty like intense
file = '5th Symphony Metal Version.mp3'
pygame.init()
pygame.mixer.init()
pygame.mixer.music.load(file)
pygame.mixer.music.play(-1)
sound = pygame.mixer.Sound("break3.wav")
effect = pygame.mixer.Sound("Paddle ball hit 1.wav")
screen = pygame.display.set_mode([1200, 750])

# One life, no HUD display, how far will you go? Finish the level and you will proceed by manually
inputting the settings as you advance. Difficulty ranges from Beginner to Insane
pygame.display.set_caption('Hardcore Brick Breaker')

# Mouse to dissapear so it acts like a real brick braker
pygame.mouse.set_visible(0)

# Font setup
font = pygame.font.Font(None, 50)

#Draws Sprite and the Screen Background
background = pygame.Surface(screen.get_size())
```

```
blocks = pygame.sprite.Group()
balls = pygame.sprite.Group()
allsprites = pygame.sprite.Group()


player = Player()
allsprites.add(player)


ball = Ball()
allsprites.add(ball)
balls.add(ball)


top = 2

# Setup the blocks from the first level. Note this can be challenge by setting the block count on higher difficulties if you want
blockcount = 80



# Set the number of rows, columns,position and color of the block
for row in range(10):
    for column in range(0, blockcount):
        block = Block(red, column * (block_width + 9) + 8, top)
        blocks.add(block)
        allsprites.add(block)
    top += block_height + 13

clock = pygame.time.Clock()

#Game over?
game_over = False
#Option to exit or not
exit_program = False

# Main program loop
while not exit_program:
    #FPS Counter
    clock.tick(30)
    # Fills the screen with the color given above
    screen.fill(black)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            exit_program = True
#How does it know when it's game over?
    if not game_over:
        player.update()
        game_over = ball.update()


    if game_over:
        text = font.render("Game Over", True, red)
        textpos = text.get_rect(centerx=background.get_width()/2)
        textpos.top = 300
        screen.blit(text, textpos)

#Collision between ball, brick and paddle
    if pygame.sprite.spritecollide(player, balls, False):
        effect.play()
        #If you hit the paddle even on the right angle it will bounce randomly
        diff = (player.rect.x + player.width/2) - (ball.rect.x+ball.width/2)
        #Don't ever try to hit the paddle on the edge other wise it will be hard to bounce back
        ball.rect.y = screen.get_height() - player.rect.height - ball.rect.height - 1
        ball.bounce(diff)
```

```
    deadblocks = pygame.sprite.spritecollide(ball, blocks, True)

    if len(deadblocks) > 0:
        sound.play()
        ball.bounce(0)
        print("hit")
        # To make sure that the brick is hit and is scored
        if len(blocks) == 0:
            game_over = True
            print("Level completed")
            # Level completed and an option to go for a new challenge
            # Please note that HUD for the score is OFF
    allsprites.draw(screen)
    pygame.display.flip()

pygame.quit()



#If you quit or game over calculate the final score from the current level and the score of the previous level if you want to get challenged
Final = input("Game over score")#Total of hits on the command from the current level and the level finished previously(Remember what score is on the previous one but only for those who want to get challenged)
print(Final + "\nGame over")
```

Sources:

http://programarcadegames.com/python_examples/show_file.php?file=breakout_simple.py

https://www.pygame.org/docs/ref/music.html

https://nebelprog.wordpress.com/2013/08/14/create-a-simple-game-menu-with-pygame-pt-1-writing-the-menu-options-to-the-screen/

https://github.com/

https://freesound.org/