Odd Semester (2023)



**BINUS UNIVERSITY**

**BINUS INTERNATIONAL**

**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | |  | |  | |  |
| **Student Information**: **Surname** | | | | | | **Given Names**    **Jocelyn** | | **Student ID Number**  **2301900454** | |  |
| 1. | | | **Thiojaya** |  | |  |
|  |  | |  |
| **Course Code** | **: COMP6502** | |  |  | | **Course Name** | | **: Introduction to Programming** | |  |
| **Class** | **: L1AC** | |  |  | | **Name of Lecturer(s)** | | **: Ida Bagus Kerthyayana Manuaba** | |  |
|  |  | |  |  | |  | |  | |  |
| **Major** | **: Comp. Science** | |  |  | |  | |  | |  |
| **Title of Assignment**  (if any) | : Brick Breaker Game | | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  | |  | |  | |  | |  |
| **Due Date** | **: 17-01-20** |  | |  | | **Submission Date** | | **: 17-01-20** | |  |

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. Jocelyn Thiojaya

**“Brick Breaker Game”**

**Name : Jocelyn Thiojaya**

**ID : 2301900454**

1. **PROJECT SPECIFICATIONS**
2. **Introduction**

Brick breaker is a game involving a bat (or sometimes called a paddle), a ball, and some bricks. The ball will break bricks that it touches. The goal is to break all the bricks using the ball. The player needs to keep the ball bouncing around the play area, if it does fall to the bottom, the player loses a life. The player catches and bounces the ball back up using the bat, which is a rectangle that can be moved left and right. The player earns score for every brick that they break. When the player has no lives left, the game ends.

1. **Inspiration**

I was inspired by the blackberry game of the same name, “Brick Breaker”. It is a childhood game that almost everyone that ever had a blackberry phone must have played in some time. Now, it is hard to find a game of that sort, as it is not trending anymore.

1. **Features**

My version is going to be a game with the same essentials, which has a bat in the bottom of the screen that can be moved left and right using arrow keys, a ball that bounces off the 4 walls of the screen and all the objects in the game, and some bricks that can be broken using the ball and give the player points.

Additionally, I will be making an added feature to the bat, where you can hold down the space key to make it go faster. This will come useful in catching balls that are a far distance from the bat.

The bricks scale down by a set factor each time the player levels up. The high score a player gets is stored in a text file, so the data is saved.

1. **The Pygame Module**

This project will be using the Pygame module for python, which is mainly used to build games.

Pygame (the library) is a Free and Open Source python programming language library for making multimedia applications like games built on top of the excellent SDL library. Like SDL, pygame is highly portable and runs on nearly every platform and operating system. Millions of people have downloaded pygame itself, which is a whole lot of bits flying across the interwebs. (www.pygame.org/wiki/about)

1. **SOLUTION DESIGN**
2. **Class Diagrams**

These are the class diagrams for each object in the game. Each seen object has its own class.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Ball** |  | **Bat** |  | **BottomLine** |
| **+ screen**  **+ ai\_settings**  **+ image**  **+ rect**  **+ screen\_rect**  **+ rect.x**  **+ rect.y**  **+ change\_x**  **+ change\_y** |  | **+ screen**  **+ ai\_settings**  **+ width**  **+ height**  **+ image**  **+ rect**  **+ screen\_rect**  **+ rect.centerx**  **+ rect.bottom**  **+ center**  **+ moving\_right**  **+ moving\_left** |  | **+ screen**  **+ ai\_settings**  **+ image**  **+ rect**  **+ screen\_rect**  **+ rect.centerx**  **+ rect.bottom** |
| **+ update()**  **+ collision\_wall()**  **+ bat\_collision()**  **+ brick\_collision()**  **+ blitme()**  **+ center\_ball()** |  | **+ update()**  **+ blitme()**  **+ center\_bat()** |  | **+ blitme()** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Brick** |  | **Button** |  | **GameStats** |
| **+ screen**  **+ ai\_settings**  **+ width**  **+ height**  **+ image**  **+ rect** |  | **+ screen**  **+ screen\_rect**  **+ width**  **+ height**  **+ button\_color**  **+ text\_color**  **+ font**  **+ rect**  **+ rect.center** |  | **+ ai\_settings**  **+ game\_active**  **+ high\_score** |
| **+ blitme()** |  | **+ prep\_msg(msg)**  **+ draw\_button()** |  | **+ reset\_stats()** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Scoreboard** |  | **Settings** |  | **Title** |
| **+ screen**  **+ screen\_rect**  **+ ai\_settings**  **+ stats**  **+ text\_color**  **+ font** |  | **+ screen\_width**  **+ screen\_height**  **+ bg\_color**  **+ bat\_speed\_factor**  **+ bat\_width**  **+ bat\_height**  **+ ball\_limit**  **+ brick\_width**  **+ brick\_height**  **+ color\_code**  **+ brick\_color**  **+ brick\_points** |  | **+ screen**  **+ ai\_settings**  **+ image**  **+ rect**  **+ screen\_rect**  **+ rect.x**  **+ rect.y** |
| **+ prep\_score()**  **+ prep\_high\_score()**  **+ prep\_balls()**  **+ prep\_level()**  **+ show\_score()** |  | **+ level\_up()**  **+ reset\_bricks()** |  | **+ blitme()** |

1. **Flowchart**

No

No

START

Make all game objects

Start main loop for the game

stats.game is active

Yes

Ball bounces against walls, bricks, and bat. Can break bricks to earn points. Level up if all bricks are gone. Have 5 ball lives.

Player has used up all lives

Yes

END

1. **Class Explanation**

**Settings class (settings.py)**

This class stores the values that will be used across the other files, including screen, bat, ball, brick, score, and title settings. It also contains a function used for changing values when leveling up, and another to reset the bricks to its initial value when the player restarts the game. This file contains a list of colors used for the bricks too. Note: the instance that will be made for this class is called ai\_settings.

colors = [(255,0,0), (255,128,0), (255,217,25), (102,204,0), (0,77,0), (0,255,128),

    (0,128,255), (0,0,255), (234,128,255), (255,0,255), (255,0,127)]

class Settings():

    #Store all settings that i would like to access across all files

    def \_\_init\_\_(self):

        #Initialize the game's static settings.

        # Screen settings

        self.screen\_width = 1000

        self.screen\_height = 700

        self.bg\_color = (230, 230, 230)

        self.bot\_distance = 54

        # Bat settings

        self.bat\_speed\_factor = 1

        self.bat\_width = 170

        self.bat\_height = 7.5

        # Ball settings

        self.ball\_limit = 5

        # Brick settings

        self.brick\_width = 160

        self.brick\_height = 120

        self.color\_code = 0

        self.brick\_color = colors[self.color\_code]

        self.brick\_size\_factor = 0.8

        # Scoring

        self.brick\_points = 100

        # Title settings

        self.title\_x = +260

        self.title\_y = -550

    def level\_up(self):

        # sequence when player reaches a new level

        self.brick\_width \*=  self.brick\_size\_factor

        self.brick\_height \*=  self.brick\_size\_factor

        if self.color\_code == 10:

            self.color\_code = 0

            self.brick\_color = colors[self.color\_code]

        else:

            self.color\_code += 1

            self.brick\_color = colors[self.color\_code]

    def reset\_bricks(self):

        # reset brick settings to initial ones

        self.brick\_width = 160

        self.brick\_height = 120

        self.color\_code = 0

        self.brick\_color = colors[self.color\_code]

**Main objects:**

**Ball, Bat, Brick, and BottomLine classes (ball.py, bat.py, brick,py, bottomline.py)**

These classes are inherited from the pygame.Sprite class, so that collision detecting functions from pygame.Sprite can be used. We need to import pygame module first in order to call Sprite class and use the function pygame.image.load(). In each class, we initialize screen and ai\_settings, so that we can use values and functions from ai\_settings and screen.

Then, we make the image of the object. The ball and bottom line uses a loaded image, while the bat and brick are drawn pygame.Surface objects. In these classes we also position these objects. After that, we get the object’s rect.

**Example file; ball.py**

In ball.py the random module is also imported to be used in it’s functions later.

import pygame

import random

class Ball(pygame.sprite.Sprite):

    def \_\_init\_\_(self, ai\_settings, screen):

        # Call the parent class (Sprite) constructor

        super().\_\_init\_\_()

        # Initialize the ball and set its starting position.

        self.screen = screen

        self.ai\_settings = ai\_settings

        # Load the ball image and get its rect.

        self.image = pygame.image.load('images/ball.png')

        self.rect = self.image.get\_rect()

        self.screen\_rect = screen.get\_rect()

        self.rect.x = self.screen\_rect.centerx

        self.rect.y = self.screen\_rect.bottom - ai\_settings.bot\_distance

**Example file; bat.py**

import pygame

BLACK = (0,0,0)

class Brick(pygame.sprite.Sprite):

    def \_\_init\_\_(self, ai\_settings, screen):

        # Call the parent class (Sprite) constructor

        super().\_\_init\_\_()

        #Initialize the brick and set its starting position.

        self.screen = screen

        self.ai\_settings = ai\_settings

        self.width = ai\_settings.brick\_width

        self.height = ai\_settings.brick\_height

        # Pass in the color of the brick, and its x and y position, width and height.

        # Set the background color and set it to be transparent

        self.image = pygame.Surface([self.width, self.height])

        self.image.fill(BLACK)

        self.image.set\_colorkey(BLACK)

        # Draw the brick (a rectangle!)

        pygame.draw.rect(self.image, ai\_settings.brick\_color, [10, 10, self.width, self.height])

        # Get the rect of a brick.

        self.rect = self.image.get\_rect()

A blitme() function is added to draw the image to the screen. It calls pygame’s screen.blit()

    def blitme(self):

        # draws image to screen

        self.screen.blit(self.image, self.rect)

**Other objects:**

**Button, GameStats, Scoreboard, and Title classes (button,py, game\_stats,py, scoreboard.py, title.py)**

These classes do not inherit from the pygame.Sprite class, because we don’t need to use the features and functions of Sprite. We need to import pygame module too, to use features like image.load() and SysFont. In each class, we also initialize screen and ai\_settings, so that we can use values and functions from ai\_settings and screen. We also make all the other instances that we need in each class, such as color, width, height, font, et cetera according to needs.

**Example file; button.py**

import pygame.font

class Button():

    def \_\_init\_\_(self, ai\_settings, screen, msg):

        # Initialize button attributes.

        self.screen = screen

        self.screen\_rect = screen.get\_rect()

        # Set the dimensions and properties of the button.

        self.width, self.height = 200, 50

        self.button\_color = (167,6,6)

        self.text\_color = (255, 255, 255)

        self.font = pygame.font.SysFont(None, 48)

        # Build the button's rect object and center it.

        self.rect = pygame.Rect(0, 0, self.width, self.height)

        self.rect.center = self.screen\_rect.center

        # The button message needs to be prepped only once.

        self.prep\_msg(msg)

**Example file; game\_stats.py**

class GameStats():

    # Track statistics for Breakout.

    def \_\_init\_\_(self, ai\_settings):

        # Initialize statistics.

        self.ai\_settings = ai\_settings

        self.reset\_stats()

        # Start Breakout in an inactive state.

        self.game\_active = False

        # High score should never be reset.

        f = open("highscore.txt", "r")

        self.high\_score = int(f.read())

1. **DISCUSSION**
2. **How the Ball Bounces**

Instances to the ball’s x,y coordinate changes are added, and they use random numbers between -1 and 2, to get a dynamic feel when playing the game.

Function update(self) is used to update the ball’s rect position and make the ball move.

Function do\_collision\_wall(self) changes the ball direction when the ball reaches the edge of the screen, and then bounces the ball back, moving its direction to the opposite.

Function bat\_collision(self) changes the ball direction when the ball collides with the bat. Its x coordinate may move left or right, but y coordinate must go upwards.

Function brick\_collision(self) changes the ball direction when it collides with a brick, or multiple bricks. Each collision makes the ball go the opposite direction.

class Ball(pygame.sprite.Sprite):

    def \_\_init\_\_(self, ai\_settings, screen):

        --snip--

        # Credits to: http://programarcadegames.com/python\_examples/f.php?file=bouncing\_balls.py

        # Values to change x,y coordinates

        self.change\_x = random.randrange(-1, 2)

        self.change\_y = random.randrange(1, 2)

    def update(self):

        # updates the ball's rect

        self.rect.x += self.change\_x

        self.rect.y += self.change\_y

    def do\_collision\_wall(self):

        # when collide with a wall, bounces

        if self.rect.x>=self.screen\_rect.right or self.rect.x<=0:

            self.change\_x \*= -1

        if self.rect.y>self.screen\_rect.bottom or self.rect.y<0:

            self.change\_y \*= -1

    def bat\_collision(self):

        # when collide with a bat, bounces

        self.change\_x = random.randrange(-1, 2)

        self.change\_y = - self.change\_y

    def brick\_collision(self):

        # when collide with a brick, bounces

        self.change\_x \*= -1 #random.randrange(-1, 2)

        self.change\_y \*= -1 #random.randrange(-1, 2)

    def blitme(self):

        # draws image to screen

        self.screen.blit(self.image, self.rect)

    def center\_ball(self, ai\_settings):

        # Center the ball on the screen.

        self.rect.x = self.screen\_rect.centerx

        self.rect.y = self.screen\_rect.bottom - ai\_settings.bot\_distance

1. **Game Functions (game\_functions.py)**

Game functions file contain all of the main functions for the game. There are a lot of functions that we are working with, and it would be inefficient to discuss every single one here, so here are a few examples of them.

The function check\_events which checks all the key presses and key releases and responds to them.

def check\_events(ai\_settings, screen, stats, sb, bat, bricks, play\_button):

    # Respond to keypresses and mouse events.

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            sys.exit()

        elif event.type == pygame.KEYDOWN:

            check\_keydown\_events(ai\_settings, event, bat)

        elif event.type == pygame.KEYUP:

            check\_keyup\_events(ai\_settings, event, bat)

        elif event.type == pygame.MOUSEBUTTONDOWN:

            mouse\_x, mouse\_y = pygame.mouse.get\_pos()

            check\_play\_button(ai\_settings, screen, stats, sb, bat, bricks, play\_button,

                mouse\_x, mouse\_y)

The function check\_play\_button, which starts the game or resets it when the player clicks Play. It changes the game state to be active, and empties all the current objects and recreates them.

def check\_play\_button(ai\_settings, screen, stats, sb, bat, bricks, play\_button,

    mouse\_x, mouse\_y):

    # Start a new game when the player clicks Play.

    button\_clicked = play\_button.rect.collidepoint(mouse\_x, mouse\_y)

    if button\_clicked and not stats.game\_active:

        # Hide the mouse cursor.

        pygame.mouse.set\_visible(False)

        # Reset the game statistics.

        stats.reset\_stats()

        stats.game\_active = True

        # Empty the list of bricks and reset brick settings

        bricks.empty()

        ai\_settings.reset\_bricks()

        # Create a new rows of bricks and center the bat.

        create\_rowbricks(ai\_settings, screen, bricks)

        bat.center\_bat()

        # Reset the scoreboard images

        sb.prep\_score()

        sb.prep\_high\_score()

        sb.prep\_level()

        sb.prep\_balls()

The function do\_collision\_brick, which uses the function by pygame called sprite.spritecollide. This function detects whether 2 sprite objects are colliding, and then deletes the sprite in the second argument. Sprite.spritecollide takes 2 sprites as arguments, and a true or false as the third argument to determine if to kill the sprite or not.

def do\_collision\_brick(ai\_settings, stats, sb, ball, bricks):

    # sequence for collisions with brick

    if pygame.sprite.spritecollide(ball, bricks, True):

    #for brick in pygame.sprite.spritecollide(ball, bricks, True):

        ball.brick\_collision()

        stats.score += ai\_settings.brick\_points

        sb.prep\_score()

        check\_high\_score(stats, sb)

The function ball\_die, which contains the sequence for when the player loses a life. It detects if the ball has collided with the bottom line, which is positioned just below the screen. If it has collided, the player loses a life, which means they have 1 less number of balls.

In this function, it also detects if there are no more bricks left, and will change the settings to the next level, and draw the next level.

Else, if the player has no balls left, stats.game\_active will be set to false, ending the game.

def ball\_die(ai\_settings, stats, screen, sb, bat, bricks, ball, bottom\_line):

    # Respond to ball dying, aka player loses a life.

    if stats.balls\_left > 0:

        col = pygame.sprite.collide\_rect(ball, bottom\_line)

        if col == True:

            # Decrement balls\_left.

            print('hit bottom')

            stats.balls\_left -= 1

            # Update scoreboard

            sb.prep\_balls()

            # center bat and ball

            bat.center\_bat()

            ball.center\_ball(ai\_settings)

            # pause

            sleep(0.5)

        if len(bricks) == 0:

            # Destroy existing bricks

            bricks.empty()

            # center bat and ball

            bat.center\_bat()

            ball.center\_ball(ai\_settings)

            # change settings to be harder

            ai\_settings.level\_up()

            # Increase level.

            stats.level += 1

            sb.prep\_level()

            # create new smaller bricks

            create\_rowbricks(ai\_settings, screen, bricks)

    else:

        stats.game\_active = False

        pygame.mouse.set\_visible(True)

1. **Scoring**

Drawing score is all in the Scoreboard class, which is a class for all the text you see in the game. The actual score is stored in the GameStats class. The score addition function is in game\_functions, along with the check high score function.

The Scoreboard class’ purpose is only to draw the text to the screen. There are 4 scoreboards each with their own similar function:

        self.prep\_score()

        self.prep\_high\_score()

        self.prep\_balls()

        self.prep\_level()

**Example function; prep\_score()**

    def prep\_score(self):

        # Turn the score into a rendered image.

        rounded\_score = int(round(self.stats.score, -1))

        score\_str = "{:,}".format(rounded\_score)

        self.score\_image = self.font.render(score\_str, True, self.text\_color, self.ai\_settings.bg\_color)

        # Display the score at the top right of the screen.

        self.score\_rect = self.score\_image.get\_rect()

        self.score\_rect.right = self.screen\_rect.right - 20

        self.score\_rect.top = 10

**From game\_functions.py,** the function do\_collision\_brick. This is where the score gets added. See that the stats.score is being added when a brick collision is done:

def do\_collision\_brick(ai\_settings, stats, sb, ball, bricks):

    # sequence for collisions with brick

    if pygame.sprite.spritecollide(ball, bricks, True):

    #for brick in pygame.sprite.spritecollide(ball, bricks, True):

        ball.brick\_collision()

        stats.score += ai\_settings.brick\_points

        sb.prep\_score()

        check\_high\_score(stats, sb)

**From game\_stats.py,** this is where the scores are stored. See that initial score is set as 0 in the attribute self.score. High score is stored in self.high\_score, which reads a text document containing the high score. By using a text document, the high score of a player will still be stored when the game is closed.

class GameStats():

    # Track statistics for Breakout.

    def \_\_init\_\_(self, ai\_settings):

        # Initialize statistics.

        self.ai\_settings = ai\_settings

        self.reset\_stats()

        # Start Breakout in an inactive state.

        self.game\_active = False

        # High score should never be reset.

        f = open("highscore.txt", "r")

        self.high\_score = int(f.read())

    def reset\_stats(self):

        # Initialize statistics that can change during the game.

        self.balls\_left = self.ai\_settings.ball\_limit

        self.score = 0

        self.level = 1

**From game\_functions.py,** the function check\_high\_score. This is where the high score gets checked. If the current score is higher than the high score written in the text document, this function will write the new high score to the text document.

def check\_high\_score(stats, sb):

    # Check to see if there's a new high score.

    if stats.score > stats.high\_score:

        stats.high\_score = stats.score

        f = open("highscore.txt", "w")

        f.write(str(stats.high\_score))

        sb.prep\_high\_score()

1. **The Main File (breakout.py)**

This is the main file which should be run when wanting to play the game. First, we import all the modules that we need, and all the classes that we have made.

import pygame

from pygame.sprite import Group

from settings import Settings

from game\_stats import GameStats

from bat import Bat

from ball import Ball

from brick import Brick

from bottom\_line import BottomLine

from button import Button

from scoreboard import Scoreboard

from title import Title

import game\_functions as gf

Now, we initialize pygame, and make the instance ai\_settings for the class Settings. We make a screen using pygame.display.set\_mode and use the settings previously stored in the Settings class. Then, all the instances of the game objects are made.

Finally, we start the main loop for the game. The function for check\_events is first called, to detect if the player presses the Play button. It continues to check whether stats.game\_active has been set to True, which will happen when the player presses the Play button. Once set to True, all the game\_functions are called. When the player has no balls left, stats.game\_active will be set to False, and the game resets.

def run\_game():

    # Initialize pygame, settings and create a screen object.

    pygame.init()

    ai\_settings = Settings()

    screen = pygame.display.set\_mode(

        (ai\_settings.screen\_width, ai\_settings.screen\_height))

    pygame.display.set\_caption("Brick Breaker")

    # Make a bat, ball, bottomline

    bat = Bat(ai\_settings, screen)

    ball = Ball(ai\_settings, screen)

    bottom\_line = BottomLine(ai\_settings, screen)

    # Make bricks

    bricks = Group()

    gf.create\_rowbricks(ai\_settings, screen, bricks)

    # Make stats, scoreboard, title, and play button

    stats = GameStats(ai\_settings)

    sb = Scoreboard(ai\_settings, screen, stats)

    title = Title(ai\_settings, screen)

    play\_button = Button(ai\_settings, screen, "Play")

    # Start the main loop for the game.

    while True:

        # check mouse and key events

        gf.check\_events(ai\_settings, screen, stats, sb, bat, bricks, play\_button)

        if stats.game\_active:

            # sequence when player loses a life

            gf.ball\_die(ai\_settings, stats, screen, sb, bat, bricks, ball, bottom\_line)

            # sequence if ball collide with bat, and brick and wall

            if gf.is\_collision\_bat(ball, bat):

                ball.bat\_collision()

            gf.do\_collision\_brick(ai\_settings, stats, sb, ball, bricks)

            ball.do\_collision\_wall()

            # updates ball and bat position

            bat.update()

            ball.update()

        # updates the screen

        gf.update\_screen(ai\_settings, screen, stats, sb, bat, ball, bricks,

            bottom\_line, play\_button, title)

run\_game()

1. **EVIDENCE**

**A screenshot of a cell phone

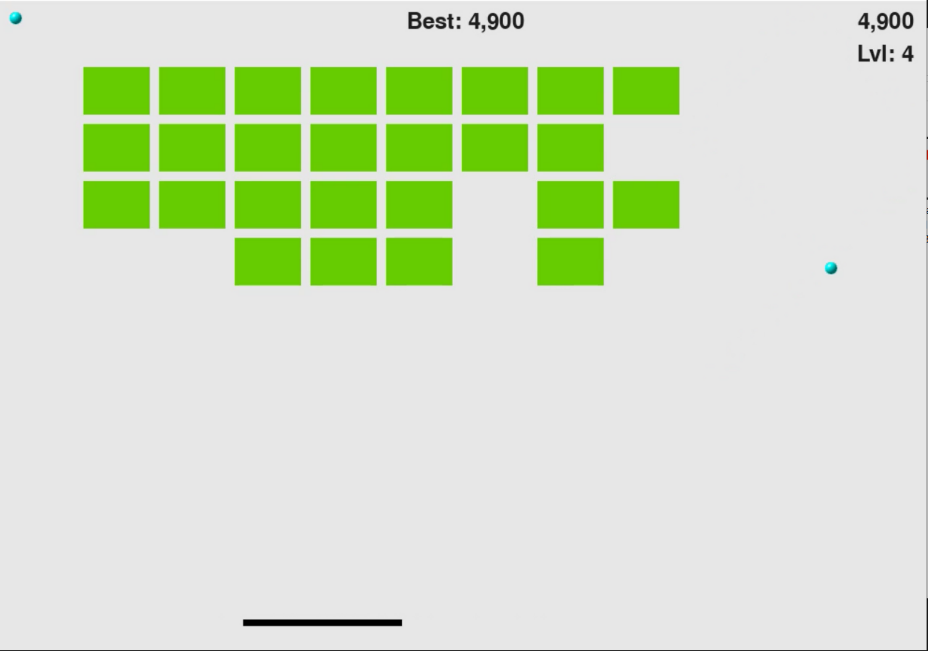
Description automatically generatedInitial starting screen:**

**A screenshot of a cell phone

Description automatically generatedLevel 1:**

**A screenshot of a cell phone

Description automatically generatedLevel 2:**

**Level 4:**

**A screenshot of a cell phone

Description automatically generatedThe player has no lives/balls left, title screen reappears:**

**A close up of a logo

Description automatically generatedThe game is reset when Play is pressed again:**

1. **REFERENCES**

**101Computing.** “**Pong Tutorial using Pygame – Adding a Bouncing Ball”.** [**https://www.101computing.net/pong-tutorial-using-pygame-adding-a-bouncing-ball/**](https://www.101computing.net/pong-tutorial-using-pygame-adding-a-bouncing-ball/)

**ProgramArcadeGames. Python Examples, bouncing\_balls.py.**

[**http://programarcadegames.com/python\_examples/f.php?file=bouncing\_balls.py**](http://programarcadegames.com/python_examples/f.php?file=bouncing_balls.py)

**Pygame.org. About – wiki.**

[**https://www.pygame.org/wiki/about**](https://www.pygame.org/wiki/about)

**Matthes, E. (May 2019).**

**Python Crash Course, 2nd Edition: A Hands-On, Project-Based Introduction to Programming**