

Algorithm and Programming (COMP6047001)
Final Project Documentation
Christoffer Raffaelo Wijaya: 2602177051
Brick-Breaker Project

Table of Contents

Overview	3
Conception.....	4
Execution.....	5
Brick Breaker Algorithm and Code	7
Reflection	13

Overview

The video game that Ali Asari made, which was preloaded at blackberry and released in 1999. It is not as exciting as other RPG or FPS game but it is an old game that i've plates since I was borned. In my era, the blackberry has only half screen and the other half is the keyboard. So basically, the game is about the player deflecting a bouncing ball with a paddle in order to break down a brick wall. The paddle can be moved horizontally and is moved with the touch of a finger, the computer mouse, or the trackwheel on a BlackBerry. A life is lost if the ball lands at the bottom of the screen, the player starts out with three lives. Because of the original Brick Breaker has a lot of levels, I only make one level so the game is not that hard. Once you move to the next level, it would be hard to break the brick. Brick Breaker comes in a variety of variations, some of which let you launch fiery fireballs or use more than one ball if the player is stuck.

Conception

Originally, I actually aimed to recreate a car game in pygame considering that I have played car games since i was little. However, with a few errors that I will go over later and a few time constraint issues, I thought that the implementation of cars, rivals for textures seemed too ambitious especially for my first semester. Nevertheless, Pygame has several features that can help with the Brick-Breaker ,such as the use of shapes and number of the bricks that could be break by the ball. By using that, I could create the ball and the brick which was to break with the use of a combination of lines and rectangles. I started my production on the 28th of November but I changed my topics due to the error that occurred in my visual studio code.

Executions

Modules:

- Pygame
 - Used different kinds of function in the program that is made in windows
- Math
 - It has always been present and is included in the standard Python release and the feature has something to do with the mathematical operations
- Sys
 - It is to exit the brick breaker game

Files: (main.py) because I only make one level

The window game which was made 600 by 800 with one entire screen that had the brick 10 times 3. It was made to be break and it is not a brick that can be hit once and broken. To make it harder, I make the brick, which has to be break twice in order for the brick to be broken. For each player that plays brick breaker game, they are given 3 lives and once the live is over, that means the game is over and you should restart again.

The player can also move the bar to left and right using the right or left chevron button on the motherboard. Although brick breaker is known one of the oldest game, it is also known as one of the most interesting game which I first try before I am into RPG, FPS and MOBA games.

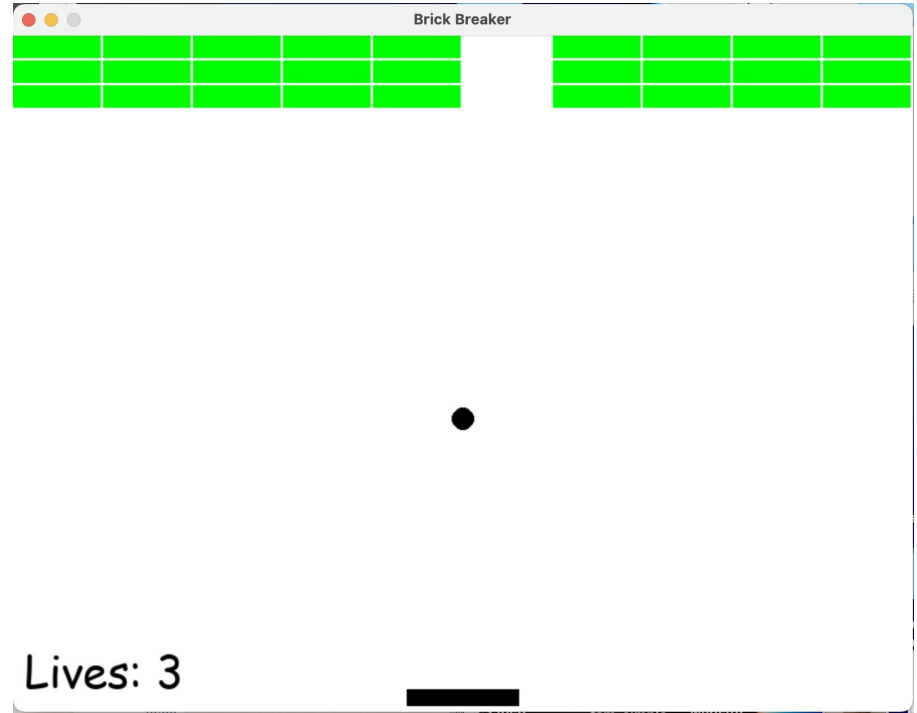


Figure 1

Brick Breaker Algorithm and Code

This code is a basic implementation of the game "Brick Breaker" using the Pygame library in Python. It contains several classes and functions that work together to create the game.

First, the code initializes Pygame and sets the display size, caption, and FPS. It uses the `pygame.init()` function to initialize the Pygame library, and the `pygame.display.set_mode()` function to set the display size to 800x600 pixels. It also uses the `pygame.display.set_caption()` function to set the caption of the game window to "Brick Breaker".

Then, the code defines three classes: Paddle, Ball, and Brick. Each class has its own properties, methods, and behavior.

The Paddle class has a `VEL` class variable, which is set to 5, and an `init` method that is used to initialize the object. This method takes in the `x` and `y` position, `width`, `height`, and `color` of the paddle as arguments. It also has a `draw` method, which is used to draw the paddle on the screen, and a `move` method which is used to move the paddle in the specified direction.

```
1  import pygame
2  import math
3
4  pygame.init()
5
6  WIDTH, HEIGHT = 800, 600
7  win = pygame.display.set_mode((WIDTH, HEIGHT))
8  pygame.display.set_caption("Brick Breaker")
9
10 FPS = 60
11 PADDLE_WIDTH = 100
12 PADDLE_HEIGHT = 15
13 BALL_RADIUS = 10
14
15 LIVES_FONT = pygame.font.SysFont("comicans", 40)
16
17
18 class Paddle:
19     VEL = 5
20
21     def __init__(self, x, y, width, height, color):
22         self.x = x
23         self.y = y
24         self.width = width
25         self.height = height
26         self.color = color
27
28     def draw(self, win):
29         pygame.draw.rect(
30             win, self.color, (self.x, self.y, self.width, self.height))
31
32     def move(self, direction=1):
33         self.x = self.x + self.VEL * direction
34
35
36 class Ball:
37     VEL = 5
```

The Ball class has a VEL class variable, which is set to 5, and an init method that is used to initialize the object. This method takes in the x and y position, radius, and color of the ball as arguments. It also has a move method that updates the ball's position based on its current velocity, a set_vel method that sets the velocity of the ball, and a draw method that is used to draw the ball on the screen.

The Brick class has an init method that takes in the x and y position, width, height, health, and colors of the brick as arguments. It has a draw method that is used to draw the brick on the screen, a collide method that detects when the ball collides with the brick, and a hit method that decrements the brick's health and changes its color based on its remaining health.

It also has a static method interpolate that takes in the color_a, color_b and the t as argument and it is used to interpolate the color of the brick based on the health.

The code also defines a draw function that takes in the window, paddle, ball, bricks, and lives as arguments. This function is used to update the display with the current state of the game. It uses the fill() function to fill the background with white color, the draw() method of the Paddle and Ball class to draw the paddle and ball on the screen, and a for loop to iterate through the list of bricks and draw each brick on the screen. It also uses the pygame.font.SysFont() function to create a font object for the text that displays the number of lives, the render() method of the font object to render the text, and the blit() function to draw the text on the screen. Finally, it uses the pygame.display.update() function to update the display.

The code also defines a ball_collision function that takes in the ball as an argument. This function is used to handle the ball's movement and collision with the walls and bricks. It checks if the ball has collided with the left or right wall, and if so, it changes the x velocity of the ball. It also checks if the ball has collided with the top or

```
39 def __init__(self, x, y, radius, color):
40     self.x = x
41     self.y = y
42     self.radius = radius
43     self.color = color
44     self.x_vel = 0
45     self.y_vel = -self.VEL
46
47 def move(self):
48     self.x += self.x_vel
49     self.y += self.y_vel
50
51 def set_vel(self, x_vel, y_vel):
52     self.x_vel = x_vel
53     self.y_vel = y_vel
54
55 def draw(self, win):
56     pygame.draw.circle(win, self.color, (self.x, self.y), self.radius)
57
58
59 class Brick:
60     def __init__(self, x, y, width, height, health, colors):
61         self.x = x
62         self.y = y
63         self.width = width
64         self.height = height
65         self.health = health
66         self.max_health = health
67         self.colors = colors
68         self.color = colors[0]
69
70 def draw(self, win):
71     pygame.draw.rect(
72         win, self.color, (self.x, self.y, self.width, self.height))
73
74 def collide(self, ball):
75     if not (ball.x <= self.x + self.width and ball.x >= self.x):
```


The code also defines a draw function that takes in the window, paddle, ball, bricks, and lives as arguments. This function is used to update the display with the current state of the game. It uses the fill() function to fill the background with white color, the draw() method of the Paddle and Ball class to draw the paddle and ball on the screen, and a for loop to iterate through the list of bricks and draw each brick on the screen. It also uses the pygame.font.SysFont() function to create a font object for the text that displays the number of lives, the render() method of the font object to render the text, and the blit() function to draw the text on the screen. Finally, it uses the pygame.display.update() function to update the display.

The code also defines a ball_collision function that takes in the ball as an argument. This function is used to handle the ball's movement and collision with the walls and bricks. It checks if the ball has collided with the left or right wall, and if so, it changes the x velocity of the ball.

```
76         return False
77     if not (ball.y - ball.radius <= self.y + self.height):
78         return False
79
80     self.hit()
81     ball.set_vel(ball.x_vel, ball.y_vel * -1)
82     return True
83
84     def hit(self):
85         self.health -= 1
86         self.color = self.interpolate(
87             *self.colors, self.health/self.max_health)
88
89     @staticmethod
90     def interpolate(color_a, color_b, t):
91         # 'color_a' and 'color_b' are RGB tuples
92         # 't' is a value between 0.0 and 1.0
93         # this is a naive interpolation
94         return tuple(int(a + (b - a) * t) for a, b in zip(color_a, color_b))
95
96
97 def draw(win, paddle, ball, bricks, lives):
98     win.fill("white")
99     paddle.draw(win)
100    ball.draw(win)
101
102    for brick in bricks:
103        brick.draw(win)
104
105    lives_text = LIVES_FONT.render(f"Lives: {lives}", 1, "black")
106    win.blit(lives_text, (10, HEIGHT - lives_text.get_height() - 10))
107
108    pygame.display.update()
109
110
111 def ball_collision(ball):
112     if ball.x - BALL_RADIUS <= 0 or ball.x + BALL_RADIUS >= WIDTH:
```

```

113     ball.set_vel(ball.x_vel * -1, ball.y_vel)
114     if ball.y + BALL_RADIUS >= HEIGHT or ball.y - BALL_RADIUS <= 0:
115         ball.set_vel(ball.x_vel, ball.y_vel * -1)
116
117
118 def ball_paddle_collision(ball, paddle):
119     if not (ball.x <= paddle.x + paddle.width and ball.x >= paddle.x):
120         return
121     if not (ball.y + ball.radius >= paddle.y):
122         return
123
124     paddle_center = paddle.x + paddle.width/2
125     distance_to_center = ball.x - paddle_center
126
127     percent_width = distance_to_center / paddle.width
128     angle = percent_width * 90
129     angle_radians = math.radians(angle)
130
131     x_vel = math.sin(angle_radians) * ball.VEL
132     y_vel = math.cos(angle_radians) * ball.VEL * -1
133
134     ball.set_vel(x_vel, y_vel)
135
136
137 def generate_bricks(rows, cols):
138     gap = 2
139     brick_width = WIDTH // cols - gap
140     brick_height = 20
141
142     bricks = []
143     for row in range(rows):
144         for col in range(cols):
145             brick = Brick(col * brick_width + gap * col, row * brick_height +
146                 gap * row, brick_width, brick_height, 2, [(0, 255, 0), (255, 0, 0)])
147             bricks.append(brick)
148
149     return bricks

```

```

152 def main():
153     clock = pygame.time.Clock()
154
155     paddle_x = WIDTH/2 - PADDLE_WIDTH/2
156     paddle_y = HEIGHT - PADDLE_HEIGHT - 5
157     paddle = Paddle(paddle_x, paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT, "black")
158     ball = Ball(WIDTH/2, paddle_y - BALL_RADIUS, BALL_RADIUS, "black")
159
160     bricks = generate_bricks(3, 10)
161     lives = 3
162
163     def reset():
164         paddle.x = paddle_x
165         paddle.y = paddle_y
166         ball.x = WIDTH/2
167         ball.y = paddle_y - BALL_RADIUS
168
169     def display_text(text):
170         text_render = LIVES_FONT.render(text, 1, "red")
171         win.blit(text_render, (WIDTH/2 - text_render.get_width() /
172             2, HEIGHT/2 - text_render.get_height()/2))
173         pygame.display.update()
174         pygame.time.delay(3000)
175
176     run = True
177     while run:
178         clock.tick(FPS)
179
180         for event in pygame.event.get():
181             if event.type == pygame.QUIT:
182                 run = False
183                 break
184
185         keys = pygame.key.get_pressed()
186
187         if keys[pygame.K_LEFT] and paddle.x - paddle.VEL >= 0:

```

```

152 def main():
153     clock = pygame.time.Clock()
154
155     paddle_x = WIDTH/2 - PADDLE_WIDTH/2
156     paddle_y = HEIGHT - PADDLE_HEIGHT - 5
157     paddle = Paddle(paddle_x, paddle_y, PADDLE_WIDTH, PADDLE_HEIGHT, "black")
158     ball = Ball(WIDTH/2, paddle_y - BALL_RADIUS, BALL_RADIUS, "black")
159
160     bricks = generate_bricks(3, 10)
161     lives = 3
162
163     def reset():
164         paddle.x = paddle_x
165         paddle.y = paddle_y
166         ball.x = WIDTH/2
167         ball.y = paddle_y - BALL_RADIUS
168
169     def display_text(text):
170         text_renderer = LIVES_FONT.render(text, 1, "red")
171         win.blit(text_renderer, (WIDTH/2 - text_renderer.get_width() /
172                                 2, HEIGHT/2 - text_renderer.get_height()/2))
173         pygame.display.update()
174         pygame.time.delay(3000)
175
176
177     run = True
178     while run:
179         clock.tick(FPS)
180
181         for event in pygame.event.get():
182             if event.type == pygame.QUIT:
183                 run = False
184                 break
185
186         keys = pygame.key.get_pressed()
187
188         if keys[pygame.K_LEFT] and paddle.x - paddle.VEL >= 0:

```

```

189         paddle.move(-1)
190     if keys[pygame.K_RIGHT] and paddle.x + paddle.width + paddle.VEL <= WIDTH:
191         paddle.move(1)
192
193     ball.move()
194     ball_collision(ball)
195     ball_paddle_collision(ball, paddle)
196
197     for brick in bricks[:]:
198         brick.collide(ball)
199
200         if brick.health <= 0:
201             bricks.remove(brick)
202
203     # lives check
204     if ball.y + ball.radius >= HEIGHT:
205         lives -= 1
206         ball.x = paddle.x + paddle.width/2
207         ball.y = paddle.y - BALL_RADIUS
208         ball.set_vel(0, ball.VEL * -1)
209
210     if lives <= 0:
211         bricks = generate_bricks(3, 10)
212         lives = 3
213         reset()
214         display_text("You Lost!")
215
216     if len(bricks) == 0:
217         bricks = generate_bricks(3, 10)
218         lives = 3
219         reset()
220         display_text("You Won!")
221
222     draw(win, paddle, ball, bricks, lives)
223
224     pygame.quit()
225     quit()

```

```

if __name__ == "__main__":
    main()

```

Reflection

Making pygame is not a easy kind of project for me in the final project because i just learn pygame this semester and since this is the first semester, maybe the brick breaker game that i made is just a simple game which is famous in the old eras and my brick breaker game only can go right and left. With this project, I learned a lot about pygame and it is really fun if we can make our own game in the future. I hope I can make more difficult games than this. The problems that I occurred at first that the code is not working but now it is working well. The buttons that we can use at the game were only right and left chevron button.