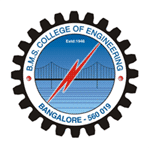
****

**B.M.S. College of Engineering, Bangalore-19**

**PUZZLE BASED LEARNING**

**“THE ROOM PAINTING PUZZLE”**

**Submitted by:**

**Amrutya- 1BM19IS010**

**Prateek Jain- 1BM18IS149**

Work done as part of the Alternative Assessment for the IVSemester Undergraduate (UG) Course Analysis and Design of Algorithms (19IS4PCADA) during the Academic Year 2019-20.

**Course Coordinator:**

**Dr. B R Shambhavi**

# Table of Contents

**Page No.**

1. **Introduction**
   1. Problem Statement…………………………………………………………………………………………………1
2. **Selection of Design Techniques(with reasoning and theory)**

2.1 Design Technique 1………………………………………………………………………………2

2.2 Design Technique 2………………………………………………………………………………3

**3. Implementation 1 and results…………………………………………………………………5-8**

**4. Implementation 2 and results…………………………………………………………………9-12**

**5. Conclusion………………………………………………………………………………………………13**

5.1 Summary

5.2 Limitations

5.3 Further Enhancements

**References…………………………………………………………………………………………..14**

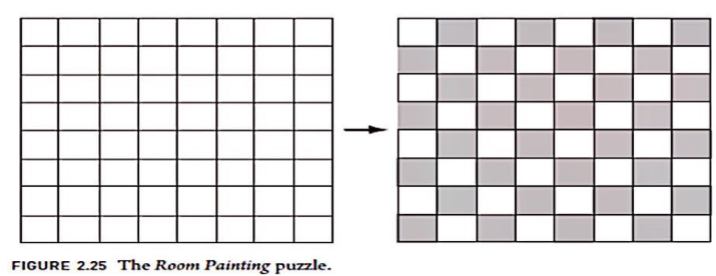
**1.INTRODUCTION**

* 1. **Problem Statement**

There once lived a king who liked chess.He had a palace whose floor plan mimicked an 8\*8 chessboard, with each of the 64 rooms having a door in each if its four walls.Originally, all the floors in the rooms were painted white. Then the king ordered the floors to be repainted so that they alternated

like the squares of a chessboard.To do this, his painter had to walk through the palace repainting

floors in all the rooms he visited from white to black, and vice versa. The painter was allowed to exit the palace and re-enter it through another door. Was there a way to execute the order by repainting the rooms not more than 60 times?

****

**2.SELECTION OF DESIGN TECHNIQUES**

* 1. **Design Technique 1**

**What is Divide And Conquer technique?**

It is an optimization of Dynamic Programming, which works by dividing the problem into sub-problems, conquer each sub-problem recursively and combine these solutions.

1. Does more work on the sub-problems and hence has more time consumption.
2. In divide and conquer the sub-problems are independent of each other

These algorithms typically solve similar pieces of the problem, and then put them together at the end. This technique is more of a recursive approach.

Divide and Conquer involves three steps at each recursion :

1. Divide the problems into sub-problems
2. Conquer the sub-problems by solving them recursively
3. Combine the solution of sub-problems into the solution for original problem
4. It is a top-down approach
5. It does more work on sub-problems and thus has more time consumption

**Why this technique?**

Notion behind the problem :

The painter enters a room, while leaving it, paints it’s floor black, not necessarily black

always.

Since symmetricity exists,the palace can be divided into equal halves diagonally

Condition is that the painter enters the room through a door and exits from the other

REQUIRED - Minimum number of times he needs to repaint all the floors

* 1. **Design Technique 2**

Dynamic programming (usually referred to as **DP** ) is a very powerful technique to solve a particular class of problems. It demands very elegant formulation of the approach and simple thinking and the coding part is very easy

 If the given problem can be broken up in to smaller sub-problems and these smaller subproblems are in turn divided in to still-smaller ones, and in this process, if you observe some over-lapping subproblems, then its a big hint for DP

There are two ways of doing this.

1. **) Top-Down :**This is referred to as ***Memoization***.
2. **2.) Bottom-Up**This is referred to as ***Dynamic Programming***.

**Why Dynamic Programming?**

Dynamic Programming enables to keep a trace of solutions to the sub-problems which reduces the operation of the same problem repeatedly over a period of time. In this puzzle question, it is useful to mark the tiles of the floor painted with either of white or black colors.

**3. Implementation 1 and results**

**import arcade  
  
SCREEN\_WIDTH = 800  
SCREEN\_HEIGHT = 600  
  
  
class MyBlock(arcade.Window):  
    """ Main application class. """  
  
    def \_\_init\_\_(self, width, height):  
        super().\_\_init\_\_(width, height)  
  
        arcade.set\_background\_color(arcade.color.AMAZON)  
  
    def setup(self):  
          
        pass  
  
    def on\_draw(self):  
          
        arcade.start\_render()  
          
  
    def update(self, delta\_time):  
          
        pass  
  
  
def main():  
    game = MyBlock(SCREEN\_WIDTH, SCREEN\_HEIGHT)  
    game.setup()  
    arcade.run()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    main()**

**def setup(self):  
   
    *# Create the sprite lists*  
    self.player\_list = arcade.SpriteList()  
    self.block\_list = arcade.SpriteList()  
  
    *# Score*  
    self.score = 0  
  
      
    self.player\_sprite = arcade.Sprite("images/character.png", SPRITE\_SCALING\_PLAYER)  
    self.player\_sprite.center\_x = 50 *# Starting position*  
    self.player\_sprite.center\_y = 50  
    self.player\_list.append(self.player\_sprite)  
  
    *# Create the block*  
    for i in range(BLOCK\_COUNT)  
          
        *# Position the block*  
        block.center\_x = random.randrange(SCREEN\_WIDTH)  
        block.center\_y = random.randrange(SCREEN\_HEIGHT)  
  
        *# Add the block to the lists*  
        self.block\_list.append(block)**

**def on\_draw(self) :**

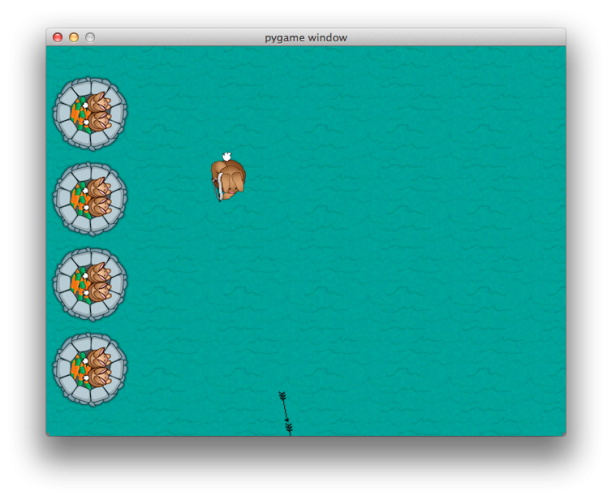
**arcade.start\_render()  
    self.block\_list.draw()  
    self.player\_list.draw()**

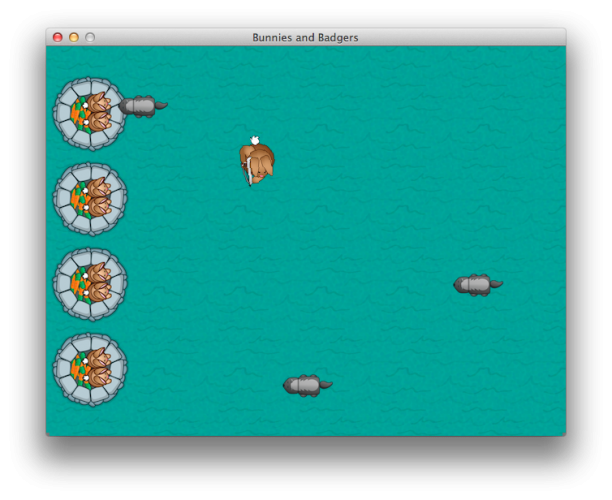
**def update(self, delta\_time):  
        block\_hit\_list = arcade.check\_for\_collision\_with\_list(self.player\_sprite, self.block\_list)**

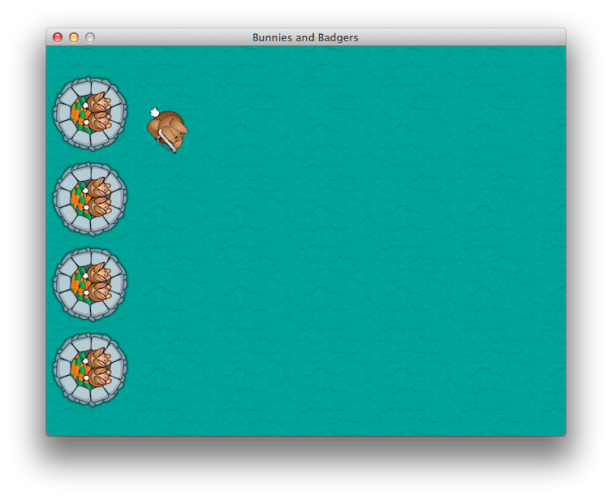
**for block in blocks\_hit\_list:  
        block.kill()  
        self.score += 1**

**MOVEMENT\_SPEED = 5  
  
def on\_key\_press(self, key, modifiers):  
      
  
    if key == arcade.key.UP:  
        self.player\_sprite.change\_y = MOVEMENT\_SPEED  
    elif key == arcade.key.DOWN:  
        self.player\_sprite.change\_y = -MOVEMENT\_SPEED  
    elif key == arcade.key.LEFT:  
        self.player\_sprite.change\_x = -MOVEMENT\_SPEED  
    elif key == arcade.key.RIGHT:  
        self.player\_sprite.change\_x = MOVEMENT\_SPEED  
  
def on\_key\_release(self, key, modifiers):  
  
  
    if key == arcade.key.UP or key == arcade.key.DOWN:  
        self.player\_sprite.change\_y = 0  
    elseif key == arcade.key.LEFT or key == arcade.key.RIGHT:  
        self.player\_sprite.change\_x = 0**

OUTPUT SNAPSHOTS:

****

****

****

****

1. **Implementation 2 and results:**

for x in range(width/grass.get\_width()+1):

for y in range(height/grass.get\_height()+1):

screen.blit(grass,(x\*100,y\*100))

screen.blit(castle,(0,30))

screen.blit(castle,(0,135))

screen.blit(castle,(0,240))

screen.blit(castle,(0,345 ))

if event.type == pygame.KEYDOWN:

if event.key==K\_w:

keys[0]=True

elif event.key==K\_a:

keys[1]=True

elif event.key==K\_s:

keys[2]=True

elif event.key==K\_d:

keys[3]=True

if event.type == pygame.KEYUP:

if event.key==pygame.K\_w:

keys[0]=False

elif event.key==pygame.K\_a:

keys[1]=False

elif event.key==pygame.K\_s:

keys[2]=False

elif event.key==pygame.K\_d:

keys[3]=False

index1=0

for bullet in arrows:

bullrect=pygame.Rect(arrow.get\_rect())

bullrect.left=bullet[1]

bullrect.top=bullet[2]

if badrect.colliderect(bullrect):

acc[0]+=1

badguys.pop(index)

arrows.pop(index1)

index1+=1

if pygame.time.get\_ticks()>=90000:

running=0

exitcode=1

if healthvalue<=0:

running=0

exitcode=0

if acc[1]!=0:

accuracy=acc[0]\*1.0/acc[1]\*100

else:

accuracy=0

# 11 - Win/lose display

if exitcode==0:

pygame.font.init()

font = pygame.font.Font(None, 24)

text = font.render("Accuracy: "+str(accuracy)+"%", True, (255,0,0))

textRect = text.get\_rect()

textRect.centerx = screen.get\_rect().centerx

textRect.centery = screen.get\_rect().centery+24

screen.blit(gameover, (0,0))

screen.blit(text, textRect)

else:

pygame.font.init()

font = pygame.font.Font(None, 24)

text = font.render("Accuracy: "+str(accuracy)+"%", True, (0,255,0))

textRect = text.get\_rect()

textRect.centerx = screen.get\_rect().centerx

textRect.centery = screen.get\_rect().centery+24

screen.blit(youwin, (0,0))

screen.blit(text, textRect)

while 1:

for event in pygame.event.get():

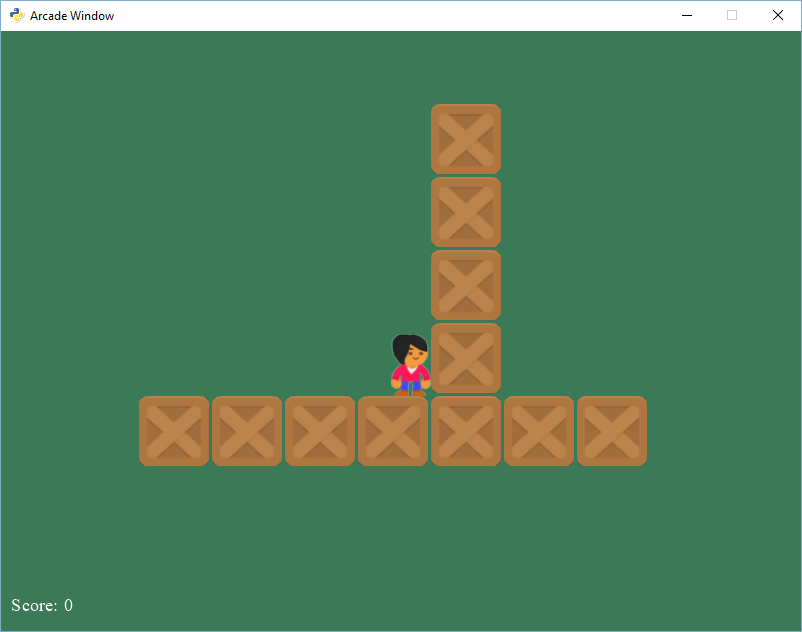
if event.type == pygame.QUIT:

pygame.quit()

exit(0)

pygame.display.fli

OUTPUT SNAPSHOTS:

****

****

**5. Conclusion**

**5.1 Summary**

**Our program code depicts the, way the painter enters the room is by pygame using arcade library using sprites, which are depicted as blocks here. Every time a collision happens, the floor is imagined to have repainted with the other color(here in the game, color change occurs). This way, alternated black and white tiles in the palace are obtained.**

**5.2 Limitations**

**This method works only when the size of 2D array is fixed, considering the minimum requirements of repainting,i.e. no more than 6o times, it turns out to be difficult to meet this requirement when the size of the 2D array (in the form of rooms of palaces here) exceeds 64. This program in python is limited only to the production of sprites and has got nothing to do with the visualization of painter painting the tiles. Another point is that this Pycode is limited to specific libraries, in this case, just the arcade.**

**5.3 Further Enhancements**

The solution for this is limited to just the sprites here and needs to be implemented further to describe the motion of the tiles changing its colors alternatively between black and white. Code needs to be improved in the aspect of its implementation.

**References**

1. **Python arcade library**
2. **Pycharm**
3. [**https://opensource.com/article/18/4/easy-2d-game-creation-python-and-arcade**](https://opensource.com/article/18/4/easy-2d-game-creation-python-and-arcade)
4. **https://www.raywenderlich.com/2795-beginning-game-programming-for-teens-with-python**