Python Project Final Report

Logic with Cute Cube

13006107 Introduction to Computers and Programming

Software Engineering Program

Faculty of Engineering, KMITL

By

63011333 Tanatapanun Pongkemmanun

Project Description:

I found that many students in my class suffer when study introduction to logic. I create this project to make propositional logic be touchable for everyone. I got inspiration from my favorite movie Wall-E and coding game 7 billion Humans to create game that can pick up and place block combined with propositional logic knowledge. You can control my cute cube by using arrow key in your keyboard. You can also pickup block and place block by press C and press P. To insert the block with logic symbol, you have to place logic block to insert space (I symbol space) in bottom right. To get the block that you insert you can press c in Delete space (D symbol space) you will get the last block that you insert to the insert space. To Test the logic that you insert, by pressing A you will get the answer next to question mark symbol. To reset all the block, you can press R. I hope this project will be useful for student that will study propositional logic.

Code:

```
import time
import pygame
import sys
#intialize the pygame
pygame.init()
pygame.display.set_caption("Logic_with_cute_cube")
#create screen
screen = pygame.display.set_mode((800,600))

#import picturcce
a = pygame.image.load("true.png").convert_alpha()
b = pygame.image.load("false.png").convert_alpha()
non = pygame.image.load("non.png").convert_alpha()
character = pygame.image.load("newc2.png").convert_alpha()
if_then = pygame.image.load("if_then.png").convert_alpha()
if_and_only_if = pygame.image.load("if_and_only_if.png").convert_alpha()
bg = pygame.image.load("bg.png").convert_alpha()
or_pict = pygame.image.load("or.png").convert_alpha()
and_pict = pygame.image.load("and.png").convert_alpha()
input_pict = pygame.image.load("input.png").convert_alpha()
delete_pict = pygame.image.load("delete.png").convert_alpha()
none_pict = pygame.image.load("none.png").convert_alpha()
notthing_pict = pygame.image.load("notthing.png").convert_alpha()

#icon
pygame.display.set_icon(character)

#Player class
class Player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.Surface([60,60])
        self.image.blit(character, (0,0))
```

```
        self.rect = self.image.get_rect()
        self.rect.centerx = 360
        self.rect.centery = 500
        self.x_change = 1
        self.y_change = 1
        self.r = 0
        self.d = 0
        self.l = 0
        self.u = 0
        self.carry = []
    def update(self):
        self.image.blit(character, (0,0))
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))
    def movement(self, game):
        if self.rect.centerx < 0:
            self.rect.centerx = 0
        if self.rect.centerx > 740:
            self.rect.centerx = 740
        if self.rect.centery < 195:
            self.rect.centery = 195
        if self.rect.centery > 540:
            self.rect.centery = 540
        if(self.r == 1):
            self.rect.centerx += self.x_change
            for block in self.carry:
                block.rect.centerx = self.rect.centerx + 10
                block.rect.centery = self.rect.centery - 19
            time.sleep(0.001)
        if(self.l == 1):
            self.rect.centerx -= self.x_change
            for block in self.carry:
                block.rect.centerx = self.rect.centerx + 10
                block.rect.centery = self.rect.centery - 19
            time.sleep(0.001)
        if(self.d == 1):
            self.rect.centery += self.y_change
            for block in self.carry:
                block.rect.centerx = self.rect.centerx + 10
                block.rect.centery = self.rect.centery - 19
            time.sleep(0.001)
        if(self.u == 1):
            self.rect.centery -= self.y_change
            for block in self.carry:
                block.rect.centerx = self.rect.centerx + 10
                block.rect.centery = self.rect.centery - 19
            time.sleep(0.001)

#class block
```

```
class Block(pygame.sprite.Sprite):
    def __init__(self, picture):
        super().__init__()
        self.image = pygame.Surface([40,40])
        self.image.blit(picture, (0,0) )
        self.rect = self.image.get_rect()


#class Block A
class True_block(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.logic = True
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class False_block(Block):
    def __init__(self, picture, x ,y):
        super().__init__(picture)
        self.logic = False
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class Non(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.connective = 2
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class If_then(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.connective = 5
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class If_and_only_if(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.connective = 6
```

```
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class And(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.connective = 3
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class Or(Block):
    def __init__(self, picture, x, y):
        super().__init__(picture)
        self.connective = 4
        self.rect.centerx = x
        self.rect.centery = y
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class Insert(Block):
    def __init__(self, picture):
        super().__init__(picture)
        self.rect.centerx = 30
        self.rect.centery = 520
        self.lst = []
        self.list_translate = []
    def get_ans(self):
        self.list_translate = []
        for x in self.lst:
            if x in connective_list:
                self.list_translate.append(x.connective)
            elif x in logic_list:
                self.list_translate.append(x.logic)
        print(self.list_translate)
        if(len(self.list_translate) == 0):
            return None
        if(len(self.list_translate) > 1):
            if( 2 not in self.list_translate and 3 not in self.list_translate and 4 not in self.list_t
ranslate and 5 not in self.list_translate and 6 not in self.list_translate):
                return None
        if(self.list_translate[0] == 3 or self.list_translate[0] == 4 or self.list_translate[0] == 5 o
r self.list_translate[0] == 6 ):
            return None
        if(type(self.list_translate[-1]) == int):
```

```
            return None
        while(len(self.list_translate) > 1):
            while(2 in self.list_translate):
                for x in range(len(self.list_translate)):
                    if(self.list_translate[x] == 2):
                        if(type(self.list_translate[x + 1]) == bool):
                            self.list_translate[x + 1] = not self.list_translate[x + 1]
                            self.list_translate.pop(x)
                            break
                        else:
                            return None
            while(3 in self.list_translate):
                for x in range(len(self.list_translate)):
                    if(self.list_translate[x] == 3):
                        if(type(self.list_translate[x-
1]) == bool and type(self.list_translate[x+1]) == bool ):
                            self.list_translate[x-1]  = self.list_translate[x-
1] and self.list_translate[x+1]
                            self.list_translate.pop(x)
                            self.list_translate.pop(x)
                            break
                        else:
                            return None
            while(4 in self.list_translate):
                for x in range(len(self.list_translate)):
                    if(self.list_translate[x] == 4):
                        if(type(self.list_translate[x-
1]) == bool and type(self.list_translate[x+1]) == bool ):
                            self.list_translate[x-1]  = self.list_translate[x-
1] or self.list_translate[x+1]
                            self.list_translate.pop(x)
                            self.list_translate.pop(x)
                            break
                        else:
                            return None
            while(5 in self.list_translate):
                for x in range(len(self.list_translate) -1, -1, -1):
                    if(self.list_translate[x] == 5):
                        if(type(self.list_translate[x-
1]) == bool and type(self.list_translate[x+1]) == bool ):
                            self.list_translate[x-1]  = not self.list_translate[x-
1] or self.list_translate[x+1]
                            self.list_translate.pop(x)
                            self.list_translate.pop(x)
                            break
                        else:
                            return None
            while(6 in self.list_translate):
```

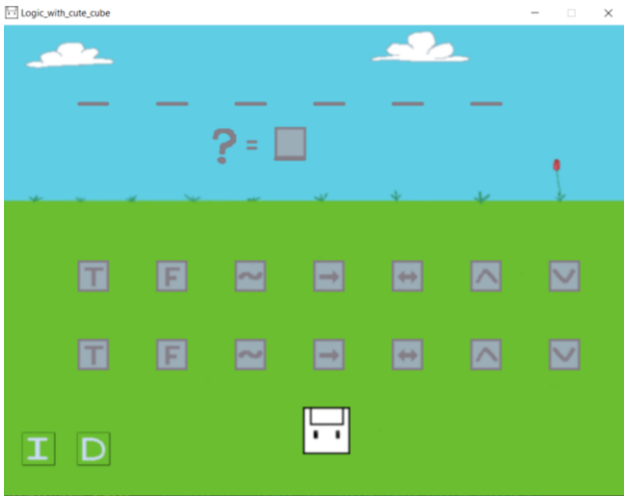
```
                for x in range(len(self.list_translate) -1, -1, -1):
                    if(self.list_translate[x] == 6):
                        if(type(self.list_translate[x-
1]) == bool and type(self.list_translate[x+1]) == bool ):
                            if(self.list_translate[x-1] == self.list_translate[x+1]):
                                self.list_translate[x-1] = True
                            else:
                                self.list_translate[x-1] = False
                            self.list_translate.pop(x)
                            self.list_translate.pop(x)
                            break
                        else:
                            return None
            if(len(self.list_translate) > 1):
                return None
        return self.list_translate[0]
    def update(self):
        for x in range(len(self.lst)):
            self.lst[x].rect.centerx = 100 * x + 100
            self.lst[x].rect.centery = 60
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))

class Pop(Block):
    def __init__(self, picture):
        super().__init__(picture)
        self.rect.centerx = 100
        self.rect.centery = 520
    def get_block(self, input_block, player):
        if(len(input_block.lst) == 0):
            return 0
        if(len(player.carry) == 0):
            player.carry.append(input_block.lst[-1])
            input_block.lst.pop(-1)
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))
class Result(Block):
    def __init__(self, picture):
        super().__init__(picture)
        self.rect.centerx = 351
        self.rect.centery = 130
    def update(self):
        screen.blit(self.image, (self.rect.centerx, self.rect.centery))
#Game class
class Game:
    def __init__(self):
        self.game_over = False
    def control(self, player):
        for event in pygame.event.get():
```

```
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_LEFT:
                    player.l = 1
                if event.key == pygame.K_RIGHT:
                    player.r = 1
                if event.key == pygame.K_UP:
                    player.u = 1
                if event.key == pygame.K_DOWN:
                    player.d = 1
                if event.key == pygame.K_c:
                    if pygame.sprite.collide_rect(player, delete_channel):
                        if(len(input_channel.lst) > 0):
                            if(len(player.carry) == 0):
                                delete_channel.get_block(input_channel, player)
                                player.carry[0].rect.centerx = player.rect.centerx + 10
                                player.carry[0].rect.centery = player.rect.centery - 19
                                player.carry[0].update()
                    elif len(player.carry) == 0:
                        collide = pygame.sprite.spritecollide(player, block_list, False)
                        if len(collide) > 0:
                            player.carry.append(collide[0])
                            player.carry[0].rect.centerx = player.rect.centerx + 10
                            player.carry[0].rect.centery = player.rect.centery - 19
                            player.carry[0].update()

                if event.key == pygame.K_p:
                    if len(player.carry) > 0:
                        player.carry[0].rect.centerx = player.rect.centerx + 10
                        player.carry[0].rect.centery = player.rect.centery + 40
                        if(len(input_channel.lst) < 6):
                            in_block_list = pygame.sprite.spritecollide(input_channel, block_list, Fal
se)

                            input_channel.lst.extend(in_block_list)
                    player.carry = []
                if event.key == pygame.K_a:
                    if input_channel.get_ans() == True:
                        result.image = a
                        result.update()
                    if input_channel.get_ans() == False:
                        result.image = b
                        result.update()
                    if input_channel.get_ans() == None:
                        result.image = none_pict
                        result.update()
                if event.key == pygame.K_r:
```

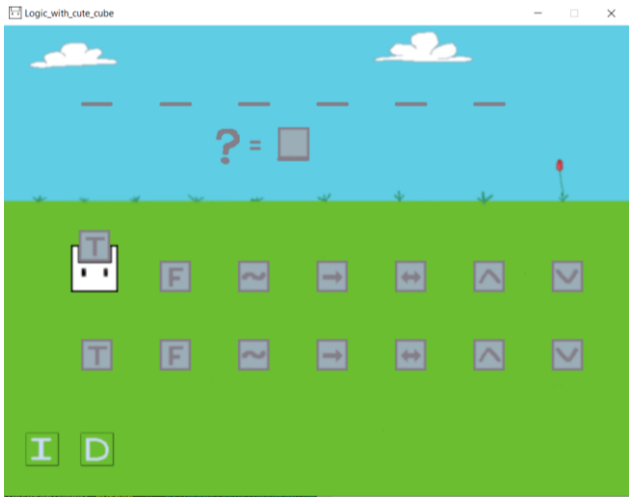
```
                    player.carry = []
                    input_channel.lst = []
                    true_block_1.rect.centerx ,true_block_1.rect.centery  = 100, 300
                    true_block_2.rect.centerx ,true_block_2.rect.centery = 100, 400
                    false_block_1.rect.centerx, false_block_1.rect.centery = 200 , 300
                    false_block_2.rect.centerx, false_block_2.rect.centery =  200 , 400
                    non_1.rect.centerx, non_1.rect.centery = 300, 300
                    non_2.rect.centerx, non_2.rect.centery = 300, 400
                    if_then_block_1.rect.centerx, if_then_block_1.rect.centery = 400, 300
                    if_then_block_2.rect.centerx, if_then_block_2.rect.centery = 400, 400
                    if_and_only_if_block_1.rect.centerx,if_and_only_if_block_1.rect.centery =  500, 30
0
                    if_and_only_if_block_2.rect.centerx,if_and_only_if_block_2.rect.centery = 500, 400
                    and_block_1.rect.centerx, and_block_1.rect.centery =  600, 300
                    and_block_2.rect.centerx, and_block_2.rect.centery =  600, 400
                    or_block_1.rect.centerx, or_block_1.rect.centery =  700, 300
                    or_block_2.rect.centerx, or_block_2.rect.centery =  700, 400
                    for x in block_list:
                        x.update()
            if event.type == pygame.KEYUP:
                if event.key == pygame.K_LEFT:
                    player.l = 0
                if event.key == pygame.K_RIGHT:
                    player.r = 0
                if event.key == pygame.K_DOWN:
                    player.d = 0
                if event.key == pygame.K_UP:
                    player.u = 0
#declare Object
player = Player()
game = Game()
true_block_1 = True_block(a, 100, 300)
true_block_2 = True_block(a, 100, 400)
false_block_1 = False_block(b, 200 , 300)
false_block_2 = False_block(b, 200 , 400)
non_1 = Non(non, 300, 300)
non_2 = Non(non, 300, 400)
if_then_block_1 = If_then(if_then, 400, 300)
if_then_block_2 = If_then(if_then, 400, 400)
if_and_only_if_block_1 = If_and_only_if(if_and_only_if, 500, 300)
if_and_only_if_block_2 = If_and_only_if(if_and_only_if, 500, 400)
and_block_1 = And(and_pict, 600, 300)
and_block_2 = And(and_pict, 600, 400)
or_block_1 = Or(or_pict, 700, 300)
or_block_2 = Or(or_pict, 700, 400)
input_channel = Insert(input_pict)
delete_channel = Pop(delete_pict)
result = Result(notthing_pict)
```

```
#create Group
connective_list = pygame.sprite.Group()
connective_list.add(non_1)
connective_list.add(non_2)
connective_list.add(if_then_block_1)
connective_list.add(if_then_block_2)
connective_list.add(if_and_only_if_block_1)
connective_list.add(if_and_only_if_block_2)
connective_list.add(and_block_1)
connective_list.add(and_block_2)
connective_list.add(or_block_1)
connective_list.add(or_block_2)
logic_list = pygame.sprite.Group()
logic_list.add(true_block_1)
logic_list.add(true_block_2)
logic_list.add(false_block_1)
logic_list.add(false_block_2)
channel_list = pygame.sprite.Group()
channel_list.add(input_channel)
channel_list.add(delete_channel)
block_list = pygame.sprite.Group()
block_list.add(true_block_1)
block_list.add(true_block_2)
block_list.add(false_block_1)
block_list.add(false_block_2)
block_list.add(non_1)
block_list.add(non_2)
block_list.add(if_then_block_1)
block_list.add(if_then_block_2)
block_list.add(if_and_only_if_block_1)
block_list.add(if_and_only_if_block_2)
block_list.add(and_block_1)
block_list.add(and_block_2)
block_list.add(or_block_1)
block_list.add(or_block_2)
block_list.add(result)
#game loop
while True:
    screen.fill((0,0,0))
    screen.blit(bg, (0,0))
    for x in channel_list:
        x.update()
    player.update()
    for x in block_list:
        x.update()
    game.control(player)
    player.movement(game)
    pygame.display.update()
```

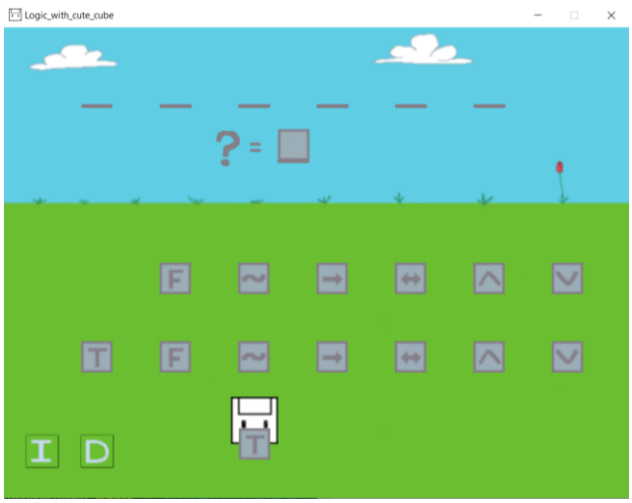
Screen Capture:



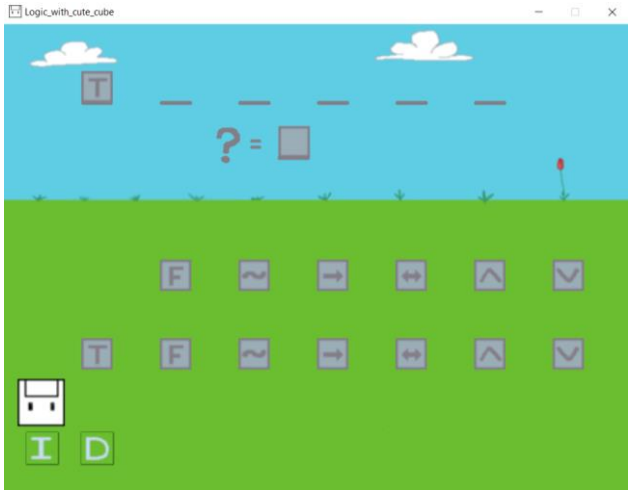
Starting page



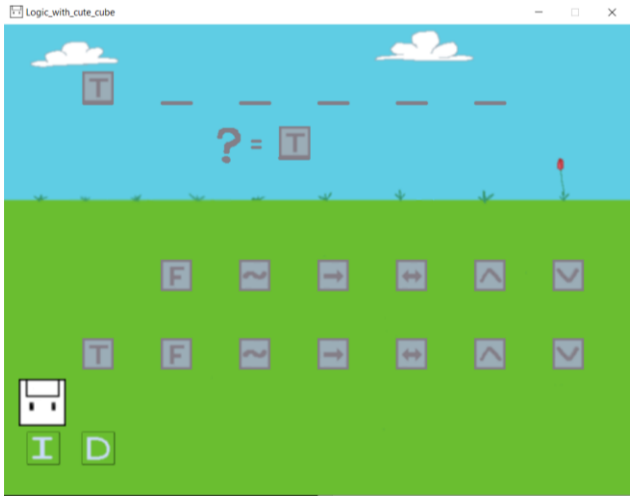
Pick up block by pressing C



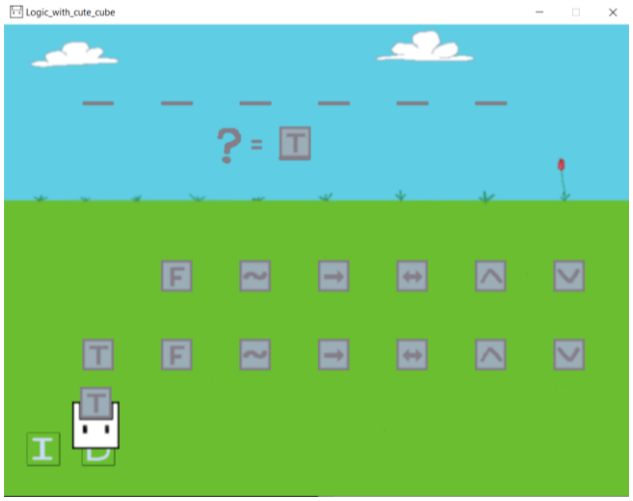
Placing block by pressing P



Placing block in Insert space



Pressing A for the answer



Pressing C in Delete space to get block in insert space