Final Project Report

Title: Klevis's Halloween Escape

By: Klevis Todi

Description: A small mini game, where you try to get the hostage outside of the ghost and goblin rooms. You also have mini tasks you need to complete before you can fully exit from the door.

Explanation: In the code I created classes which I would use to create the objects in the game (Main character, ghost, squishy monster). These objects are subclasses of the parent class "Actor", Which has a non-implanted method later implanted into these subclasses. To create the map, I created a text file with symbols on it and each symbol resembles and object (wall, door, etc.). In the text files for map on each level "X" represents a brick of wall. To create the map I initialized the game and looped through the file and created an object at the location of each symbol and attached an image onto the object. During the game there is a continuous loop running in the background. The loop ends when the player has died, and then it results into a terminating the program.

Guide: Use your keyboards arrows to navigate your character outside or the room safely, before the monsters eat him. Before any doors open you must complete each task of the levels. Level 1 you must collect all the starts. Level 2 you must box up all the mini monsters before escaping. Lastly, in level 3 you must collect each key for each door before escaping the main door.

Reflection: Overall I really enjoyed this project. It wasn't your typical college project, it really provoked your creativity, made you think outside the box, and do further learning to better your understanding in python. Even though I came across many obstacles these few weeks on this project, it greatly benefited me. These setbacks helped me analyze code better and taught me a

lot of patience. I plan on extending and building onto this game in the future. Overall I really enjoyed this semester and can't wait to learn more about coding.

Appendix:

MAIN:

```
#
"""

Created on Sun Nov 20 17:01:30 2020

@author: Todi
"""

"""Author Klevis Todi
Klevis's Halloween Escape!
University Of Michigan-Dearborn
Professor Guo
ISM 301
Final Project
"""


from game2 import Game

if __name__ == "__main__":

    game = Game()
    game.on_execute()



"""
Thank you for playing my first ever game!
Hope you enjoyed!
"""
```

ACTORS:

```
from __future__ import annotations
import pygame
from typing import Optional
from settings import *

class Actor:

    x: int
    y: int
    icon: pygame.Surface

    def __init__(self, icon_file, x, y):
        self.x, self.y = x, y
        self.icon = pygame.image.load(icon_file)

    def move(self, game: 'Game') -> None:

        raise NotImplementedError


class Player(Actor):
    x: int
    y: int
    icon: pygame.Surface
    _stars_collected: int
    _last_event: Optional[int]
    _smooth_move: bool
```

```
    _keys_collected: int

    def __init__(self, icon_file: str, x: int, y: int) -> None:

        super().__init__(icon_file, x, y)
        self._stars_collected = 0
        self._last_event = None
        self._smooth_move = False
        self._keys_collected = 0

    def set_smooth_move(self, status: bool) -> None:

        self._smooth_move = status

    def get_star_count(self) -> int:
        return self._stars_collected

    def register_event(self, event: int) -> None:
        self._last_event = event

    def get_key_count(self) -> int:
        return self._keys_collected

    def move(self, game: 'Game') -> None:
        evt = self._last_event

        if self._last_event:
            dx, dy = 0, 0
            if self._smooth_move:

                if not type(game.get_actor(self.x - 1, self.y)) == Wall:
                    if game.keys_pressed[pygame.K_LEFT]:
                        dx -= 1
                if not type(game.get_actor(self.x + 1, self.y)) == Wall:
                    if game.keys_pressed[pygame.K_RIGHT]:
                        if isinstance(game.get_actor(self.x + 1, self.y), Door):
                            if self.get_star_count() < game.goal_stars:
                                print("Door won't open unless you collect"
                                      " enough stars")
                            elif self.get_star_count() >= game.goal_stars:
                                dx += 1
                                game.game_won()
                        else:
                            dx += 1
                if not type(game.get_actor(self.x, self.y - 1)) == Wall:
                    if game.keys_pressed[pygame.K_UP]:
                        dy -= 1
                if not type(game.get_actor(self.x, self.y + 1)) == Wall:
                    if game.keys_pressed[pygame.K_DOWN]:
                        dy += 1

                if not type(game.get_actor(self.x - 1, self.y)) == Wall:
                    if game.keys_pressed[pygame.K_a]:
                        dx -= 1
                if not type(game.get_actor(self.x + 1, self.y)) == Wall:
                    if game.keys_pressed[pygame.K_d]:
                        if isinstance(game.get_actor(self.x + 1, self.y), Door):
                            if self.get_star_count() < game.goal_stars:
                                print("Door won't open unless you collect "
                                      "enough stars")
                            elif self.get_star_count() >= game.goal_stars:
                                dx += 1
                                game.game_won()
                        else:
                            dx += 1
                if not type(game.get_actor(self.x, self.y - 1)) == Wall:
                    if game.keys_pressed[pygame.K_w]:
                        dy -= 1
                if not type(game.get_actor(self.x, self.y + 1)) == Wall:
                    if game.keys_pressed[pygame.K_s]:
```

```
                dy += 1

            new_x, new_y = self.x + dx, self.y + dy

            if type(game.get_actor(new_x, new_y)) == Star:
                self._stars_collected += 1
                actor = game.get_actor(new_x, new_y)
                game.remove_actor(actor)

        else:

            if not type(game.get_actor(self.x - 1, self.y)) == Wall:
                if evt == pygame.K_LEFT or evt == pygame.K_a:
                    if game.get_level() == 1:
                        box = game.get_actor(self.x - 1, self.y)
                        if isinstance(box, Box):
                            dx -= 1
                            box.be_pushed(game, dx, dy)
                        else:
                            dx -= 1

                    if game.get_level() == 2:
                        if type(game.get_actor(self.x - 1, self.y)) == Key:
                            self._keys_collected += 1
                            actor = game.get_actor(self.x - 1, self.y)
                            game.remove_actor(actor)
                            dx -= 1
                        else:
                            dx -= 1
            if not type(game.get_actor(self.x + 1, self.y)) == Wall:
                if evt == pygame.K_RIGHT or evt == pygame.K_d:
                    if game.get_level() == 1:
                        box = game.get_actor(self.x + 1, self.y)
                        if isinstance(box, Door):
                            if game.monster_count != 0:
                                print("Door won't open unless all the"
                                      " monsters are dead")
                            elif game.monster_count == 0:
                                dx += 1
                        elif isinstance(box, Box):
                            dx += 1
                            box.be_pushed(game, dx, dy)
                        else:
                            dx += 1
                    if game.get_level() == 2:
                        if type(game.get_actor(self.x + 1, self.y)) == Key:
                            self._keys_collected += 1
                            actor = game.get_actor(self.x + 1, self.y)
                            game.remove_actor(actor)
                            dx += 1
                        elif type(game.get_actor(self.x + 1, self.y)) == Gate2:
                            if self._keys_collected == 2:
                                dx += 1
                            else:
                                print("Collect the key")
                        elif type(game.get_actor(self.x + 1, self.y)) == Door:
                            if self._keys_collected == 2:
                                dx += 1
                            else:
                                print("Collect the key")
                        else:
                            dx += 1
            if not type(game.get_actor(self.x, self.y - 1)) == Wall:
                if evt == pygame.K_UP or evt == pygame.K_w:
                    if game.get_level() == 1:
                        box = game.get_actor(self.x, self.y - 1)
                        if isinstance(box, Box):
                            dy -= 1
                            box.be_pushed(game, dx, dy)
                        else:
```

```
                    dy -= 1
                if game.get_level() == 2:
                    if type(game.get_actor(self.x, self.y - 1)) == Key:
                        self._keys_collected += 1
                        actor = game.get_actor(self.x, self.y - 1)
                        game.remove_actor(actor)
                        dy -= 1
                    elif type(game.get_actor(self.x, self.y - 1)) == Gate1:
                        if self._keys_collected == 1:
                            dy -= 1
                        else:
                            print("Collect the key")
                    else:
                        dy -= 1

            if not type(game.get_actor(self.x, self.y + 1)) == Wall:
                if evt == pygame.K_DOWN or evt == pygame.K_s:
                    if game.get_level() == 1:
                        box = game.get_actor(self.x, self.y + 1)
                        if isinstance(box, Box):
                            dy += 1
                            box.be_pushed(game, dx, dy)
                        else:
                            dy += 1
                    if game.get_level() == 2:
                        if game.get_actor(self.x, self.y + 1) == Key:
                            self._keys_collected += 1
                            actor = game.get_actor(self.x, self.y + 1)
                            game.remove_actor(actor)
                            dy += 1
                        elif type(game.get_actor(self.x + 1, self.y)) == Gate3:
                            if self._keys_collected == 2:
                                dx += 1
                            else:
                                print("Collect the key")
                        else:
                            dy += 1
            self._last_event = None

        new_x, new_y = self.x + dx, self.y + dy

        self.x, self.y = new_x, new_y
class Door(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass


class Key(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass


class Potion(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass


class Gate1(Actor):
```

```
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass


class Gate2(Actor):
    x: int
    y: int
    icon: pygame.Surface
    def move(self, game: 'Game') -> None:
        pass

class Gate3(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass

class Star(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass


class Wall(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass
class Box(Actor):
    x: int
    y: int
    icon: pygame.Surface

    def move(self, game: 'Game') -> None:
        pass

    def be_pushed(self, game: 'Game', dx: int, dy: int) -> bool:
        if type(game.get_actor(self.x, self.y)) == Box:
            if type(game.get_actor(self.x + dx, self.y + dy)) == SquishyMonster:
                game.get_actor(self.x + dx, self.y + dy).die(game)
                self.x += dx
                self.y += dy
            elif not type(game.get_actor(self.x + dx, self.y + dy)) == Wall:
                self.x += dx
                self.y += dy
                return True
        else:
            return False
class Monster(Actor):
    y: int
    icon: pygame.Surface
    _dx: float
    _dy: float
    _delay: int
    _delay_count: int

    def __init__(self, icon_file: str, x: int, y: int, dx: float, dy: float) -> None:
        super().__init__(icon_file, x, y)
        self._dx = dx
```

```
        self._dy = dy
        self._delay = 5
        self._delay_count = 1
    def move(self, game: 'Game') -> None:
        raise NotImplementedError
    def check_player_death(self, game: 'Game') -> None:
        if game.player is None:
            game.game_over()
        elif self.x == game.player.x and self.y == game.player.y:
            game.game_over()
class GhostMonster(Monster):
    x: int
    y: int
    icon: pygame.Surface
    _dx: float
    _dy: float
    _delay: int
    _delay_count: int
    movement_x: bool
    def __init__(self, icon_file: str, x: int, y: int) -> None:
        super().__init__(icon_file, x, y, 0.20, 0.20)
        self._count = 0
        self.movement_x = True
    def move(self, game: 'Game') -> None:

        if not game.player is None:
            if game.player.x > self.x:
                if (isinstance(game.get_actor(self.x + self._dx, self.y), Wall) and self.movement_x) :
                    self._count += 1
                self.x += self._dx
            if game.player.x < self.x:
                if not isinstance(game.get_actor(self.x - self._dx, self.y), Wall):
                    self.x -= self._dx

            if game.player.y > self.y:
                if not isinstance(game.get_actor(self.x, self.y + self._dy), Wall):
                    self.y += self._dy
            if game.player.y < self.y:
                if not isinstance(game.get_actor(self.x, self.y - self._dy), Wall):
                    self.y -= self._dy

            self.check_player_death(game)
class SquishyMonster(Monster):
    x: int
    y: int
    icon: pygame.Surface
    _dx: float
    _dy: float
    _delay: int
    _delay_count: int
    def __init__(self, icon_file: str, x: int, y: int) -> None:
        super().__init__(icon_file, x, y, 1, 1)

    def move(self, game: 'Game') -> None:
        if self._delay_count == 0:
            actor_obj = game.get_actor(self.x + self._dx, self.y + self._dy)
            if type(actor_obj) == Wall or type(actor_obj) == Box:
                self._dx = -1 * self._dx
                self._dy = -1 * self._dy
            else:
                self.x += self._dx
                self.y += self._dy

        self._delay_count = (self._delay_count + 1) % self._delay

        self.check_player_death(game)

    def die(self, game: 'Game') -> None:
        actor = game.get_actor(self.x, self.y)
        if type(actor) == SquishyMonster:
```

```
            game.monster_count -= 1
            game.remove_actor(actor)
```

GAME:

```
from __future__ import annotations
from typing import Optional, List
from actors2 import *
import pygame
import random

LEVEL_MAPS = ["maze1.txt", "maze3.txt", "final_maze.txt"]

def load_map(filename: str) -> List[List[str]]:

    with open(filename) as f:
        map_data = [line.split() for line in f]
    return map_data

class Game:

    def __init__(self) -> None:


        self._running = False
        self._level = 0
        self._max_level = len(LEVEL_MAPS)-1
        self.screen = None
        self.player = None
        self.keys_pressed = None

        self._actors = None
        self.stage_width, self.stage_height = 0, 0
        self.size = None
        self.goal_message = None

        self.goal_stars = 0
        self.monster_count = 0
        self.setup_current_level()


    def get_level(self) -> int:

        return self._level

    def set_player(self, player: Player) -> None:

        self.player = player

    def add_actor(self, actor: Actor) -> None:

        self._actors.append(actor)

    def remove_actor(self, actor: Actor) -> None:

        self._actors.remove(actor)

    def get_actor(self, x: int, y: int) -> Optional[Actor]:

        for i in self._actors:
            if i.x == x and i.y == y:
                return i
        return None


    def on_init(self) -> None:
        pygame.init()
        self.screen = pygame.display.set_mode \
            (self.size, pygame.HWSURFACE | pygame.DOUBLEBUF)
        self._running = True
```

```
    def on_event(self, event: pygame.Event) -> None:

        if event.type == pygame.QUIT:
            self._running = False
        elif event.type == pygame.KEYDOWN:
            self.player.register_event(event.key)

    def game_won(self) -> bool:
        obj = self.get_actor(self.player.x, self.player.y)

        if type(obj) == Door:
            if self._level == 0:
                if self.player.get_star_count() >= self.goal_stars:
                    return True
            elif self._level == 1:
                if self.monster_count == 0:
                    return True
            elif self._level == 2:
                if self.player.get_key_count >= 2:
                    return True
        else:
            return False

    def on_loop(self) -> None:
        self.keys_pressed = pygame.key.get_pressed()
        for actor in self._actors:
            actor.move(self)
        if self.player is None:
            print("You lose! :( Better luck next time.")
            self._running = False

        elif self.game_won():
            if self._level == self._max_level:
                print("Congratulations, you won!!!")
                self._running = False
            else:
                self._level += 1
                self.setup_current_level()
    def on_render(self) -> None:
        self.screen.fill(BLACK)
        for a in self._actors:
            rect = pygame.Rect(a.x * ICON_SIZE, a.y * ICON_SIZE, ICON_SIZE,
                               ICON_SIZE)
            self.screen.blit(a.icon, rect)

        font = pygame.font.Font('freesansbold.ttf', 12)
        text = font.render(self.goal_message, True, WHITE, BLACK)
        textRect = text.get_rect()
        textRect.center = (self.stage_width * ICON_SIZE // 2,
                           (self.stage_height + 0.5) * ICON_SIZE)
        self.screen.blit(text, textRect)

        pygame.display.flip()

    def on_cleanup(self) -> None:
        pygame.quit()

    def on_execute(self) -> None:
        self.on_init()

        while self._running:
            pygame.time.wait(100)
            for event in pygame.event.get():
                self.on_event(event)
            self.on_loop()
            self.on_render()

        self.on_cleanup()

    def game_over(self) -> None:
```

```
        self.player = None

    def setup_current_level(self):
        data = load_map(
            "../data/"+LEVEL_MAPS[self._level])
        if self._level == 0:
            self.setup_ghost_game(data)
        elif self._level == 1:
            self.setup_squishy_monster_game(data)
        elif self._level == 2:
            self.setup_final_maze(data)

    def setup_ghost_game(self, data) -> None:
        w = len(data[0])
        h = len(
            data) + 1

        self._actors = []
        self.stage_width, self.stage_height = w, h-1
        self.size = (w * ICON_SIZE, h * ICON_SIZE)

        player, chaser = None, None

        for i in range(len(data)):
            for j in range(len(data[i])):
                key = data[i][j]
                if key == 'P':
                    player = Player("../images/boy-24.png", j, i)
                elif key == 'C':
                    chaser = GhostMonster("../images/ghost-24.png", j, i)
                elif key == 'X':
                    self.add_actor(Wall("../images/wall-24.png", j, i))
                elif key == 'D':
                    self.add_actor(Door("../images/door-24.png", j, i))

        self.set_player(player)
        self.add_actor(player)
        player.set_smooth_move(True)
        self.add_actor(chaser)
        self.goal_stars = 5
        self.goal_message = "Objective: Collect {}".format(self.goal_stars) + \
                            " stars before the ghost gets you and head for" \
                            " the door"
        num_stars = 0
        while num_stars < 7:
            x = random.randrange(self.stage_width)
            y = random.randrange(self.stage_height)
            actors = self.get_actor(x, y)
            if not (type(actors) == Wall or type(actors) == GhostMonster or
                    type(actors) == SquishyMonster or type(actors) == Door
                    or type(actors) == Player):
                self.add_actor(Star("../images/star-24.png", x, y))
                num_stars += 1

    def setup_squishy_monster_game(self, data) -> None:

        w = len(data[0])
        h = len(
            data) + 1
        self._actors = []
        self.stage_width, self.stage_height = w, h-1
        self.size = (w * ICON_SIZE, h * ICON_SIZE)
        self.goal_message = "Objective: Squish all the monsters " \
                            "with the boxes " + " and head for the door"

        player, chaser = None, None

        for i in range(len(data)):
            for j in range(len(data[i])):
```

```
            key = data[i][j]
            if key == 'P':
                player = Player("../images/boy-24.png", j, i)
            elif key == 'M':
                self.monster_count += 1
                self.add_actor(SquishyMonster("../images/monster-"
                                              "24.png", j, i))
            elif key == 'X':
                self.add_actor(Wall("../images/wall-24.png", j, i))
            elif key == 'D':
                self.add_actor(Door("../images/door-24.png", j, i))

        self.set_player(player)
        self.add_actor(player)

        num_boxes = 0
        while num_boxes < 12:
            x = random.randrange(self.stage_width)
            y = random.randrange(self.stage_height)
            actors = self.get_actor(x, y)
            if not (type(actors) == Wall or type(
                    actors) == GhostMonster or type(
                    actors) == SquishyMonster or type(actors) == Door or type(
                    actors) == Player) or type(actors) == Box:
                self.add_actor(Box("../images/box-24.png", x, y))
                num_boxes += 1

    def setup_final_maze(self, data) -> None:
        w = len(data[0])
        h = len(
            data) + 1

        self._actors = []
        self.stage_width, self.stage_height = w, h-1
        self.size = (w * ICON_SIZE, h * ICON_SIZE)
        self.goal_message = "Objective: Collect the potion to see the keys." \
                            "Collect the keys to open each gate " + \
                            "and head for the door"

        player, chaser = None, None

        for i in range(len(data)):
            for j in range(len(data[i])):
                key = data[i][j]
                if key == 'P':
                    player = Player("../images/boy-24.png", j, i)
                elif key == 'M':
                    self.monster_count += 1
                    self.add_actor(SquishyMonster("../images/monster-"
                                                  "24.png", j, i))
                elif key == 'C':
                    chaser = GhostMonster("../images/ghost-24.png", j, i)
                elif key == 'X':
                    self.add_actor(Wall("../images/wall-24.png", j, i))
                elif key == 'D':
                    self.add_actor(Door("../images/door-24.png", j, i))
                elif key == "T":
                    self.add_actor(Potion("../images/potion-24.png", j, i))
                elif key == 'K':
                    self.add_actor(Key("../images/key-24.png", j, i))
                elif key == "G":
                    self.add_actor(Gate1("../images/door-24.png", j, i))
                elif key == "H":
                    self.add_actor(Gate2("../images/door-24.png", j, i))
        self.set_player(player)
        self.add_actor(player)
        self.add_actor(chaser)
```