Class diagram

Class name: player
Attributes: PlayerSpeed(int) (private)

Methods:
__init__(speed, x_position , y_position, width, length) (public)
player_move() (private)
draw() (public)

Class name: enemy
Attributes: EnemySpeed(int) (private)

Methods:
__init__(speed, x_position, y_position, width, length) (public)
ai_move() (private)
draw() (public)

Class name: Pushable_block
Attributes: Is_pushed(boolean)
Methods: Push(direction, sprites) #sprites is the list of things the block can collide with/be pushed by

Class name: wall

Attributes:
x_position(int) (public)
y_position(int) (public)
width(int) (public)
length(int) (public)

Methods:
__init__(x_position, y_position, width, length) (public)
draw() (public)
check_for_collision(other) (public)
get_rect() (public)

Class Diagram Explanation

The player, enemy, and pushable_block classes inherit from the wall class the following methods: draw(), check_for_collision(other), get_rect(), __init__()

The draw and get_rect methods are meant for creating the sprite, check_for_collision method checks for if a sprite has collided with another sprite.

The player_move method overrides the __init__ method and makes the code redraw the player sprite. The ai_move method overrides the __init__ method and makes the code redraw the enemy sprite. The push method overrides the __init__ method and makes the code redraw the block sprite after it has moved in the direction an enemy or player sprite has pushed it in. the check_for_collision method checks if any sprite has collided with another and stops the sprites if they do.

The methods and attributes labelled public can be inherited by a subclass, the methods and attributes labelled private can't be inherited.

Structure:
The subclasses (player, enemy, pushable box) and the main class (wall) will be in separate python files. The subclasses will import the wall class and the main file where the game loop is imports all the subclasses.

Legal and Ethical issues:
Legal and ethical issues include copyrighted materials/code and user data protection. I am not using any copyrighted code in my game and I am not using any user data.

Error handling:
For important parts of the classes, I will make it so that if an error happens, it will print a message stating where the error is, e.g. if a collision error happens (the sprites touch but don't stop each other) then a message will be printed saying which class the error is in and where/what the error is. To make sure that user input (the arrow keys) is correct, the code will print an error message if an arrow key is pressed but doesn't move the player sprite.

Testing:
I will do unit testing on each class, the classes being in separate files will help isolate them. I will do integration testing after unit testing to make sure that the classes can work together. I will do integration testing by running a test game and see what works and what changes if I change any method or attribute. The classes being in separate python files can help isolate them from each other and makes it easier to do unit testing. Unit testing will test the classes' position and movement, integration testing will test the classes' collision.

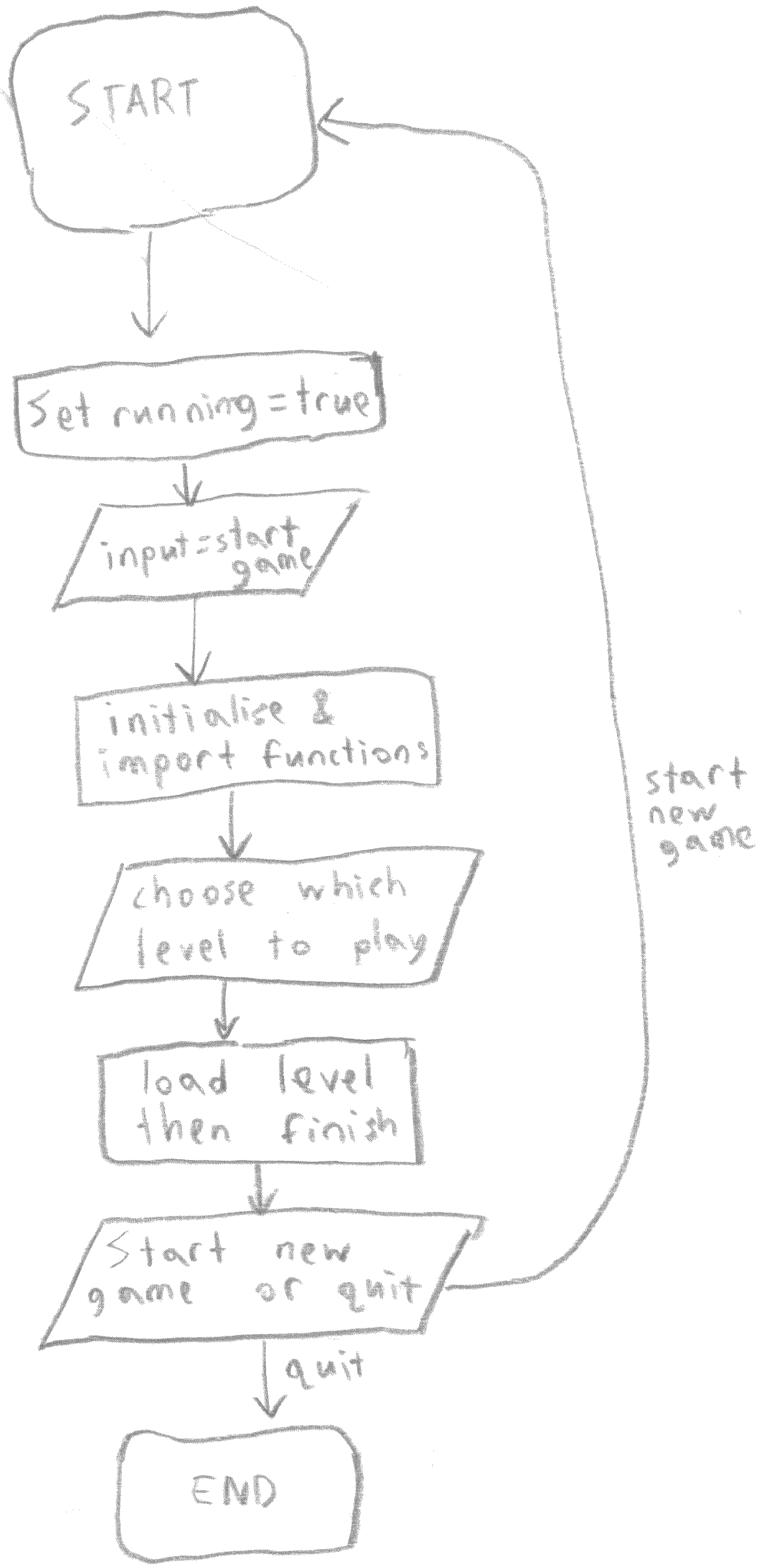
PSEUDOCODE for Wall Class (collision)

```
CLASS wall
    FUNCTION __init__(x_position, y_position, width, length)
        SET self.x_position = (position on x axis)
        SET self.y_position = (position on y axis)
        SET self.width = (how wide the wall should be)
        SET self.length = (how long the wall should be)
    FUNCTION get_rect()
        RETURN pygame.Rect(self.x_position, self.y_position, self.width,self.length) #calculates the wall to be drawn
    FUNCTION draw()
        SET rectangle TO CALL self.get_rect() #takes the above measurements and puts it into rectangle variable for use in line below
        CALL pygame.draw.rect(screen, (RGB COLOUR, E.G. 20, 20, 20), rectangle) #makes a rectangle with the colour on the screen
    FUNCTION check_for_collision(other) #function to check for collision with “other” object
        RETURN CALL self.get_rect().colliderect(other.get_rect()) #gets wall measurements and checks if overlapping another sprite, sets colliding to true if it is overlapping



#Will create separate objects as walls to put around the screen
```

PSEUDOCODE for Player Class (input checking)

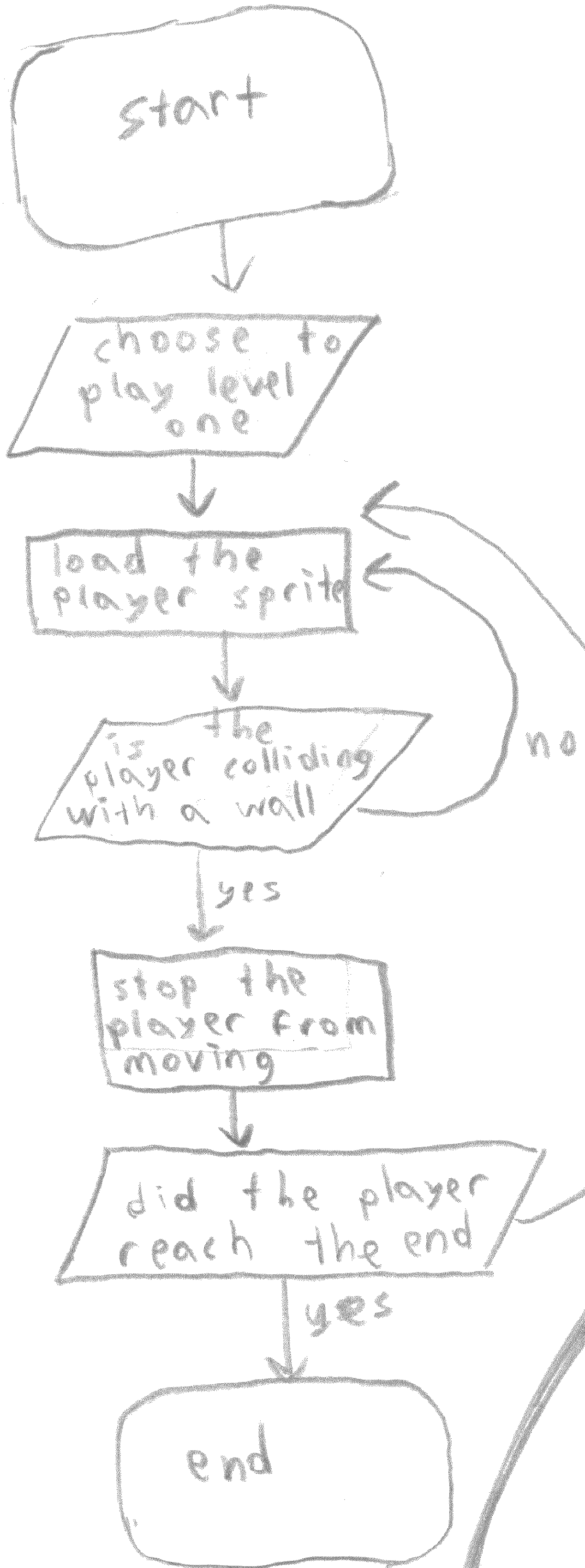
```
CLASS player
    FUNCTION __init__(x_position, y_position, width, length)
        CALL super().__init__(x_position, y_position, player_width, player_length)
        SET self.__playerspeed = speed #player speed variable, how many pixels per frame the player moves by, will change around in integration testing
        SET self.width = (how wide the player should be) #will change in unit testing
        SET self.length = (how long the player should be) #
    FUNCTION __player_move()
        GET user_input
        IF user_input == “UP”
            DECREASE self.y_position BY self.__playerspeed
        ELSE IF user_input == “DOWN”
            INCREASE self.y_position BY self.__playerspeed
        ELSE IF user_input == “LEFT”
            DECREASE self.x_position BY self.__playerspeed
        ELSE IF user_input == “RIGHT”
            INCREASE self.x_position BY self.__playerspeed
        ELSE
            PRINT “error, arrow key input went wrong” #if something goes wrong with the arrow keys, it will print the message
    FUNCTION check_for_collision(other)
        RETURN CALL super().check_for_collision(other) #inherited from the wall class
    FUNCTION draw() #inherited from wall class, edited to make it draw moving player
        CALL self.__player_move() #update player position
        CALL pygame.draw.rect(screen, (RGB COLOUR OF PLAYER SPRITE), pygame.Rect(self.x_position, self.y_position, self.width, self.length))
```

game loop flowchart

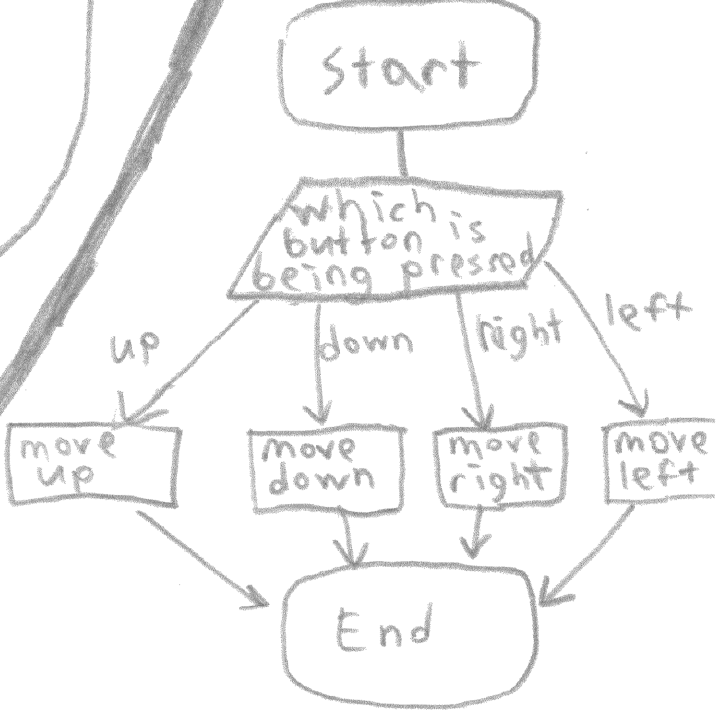


level one flowchart

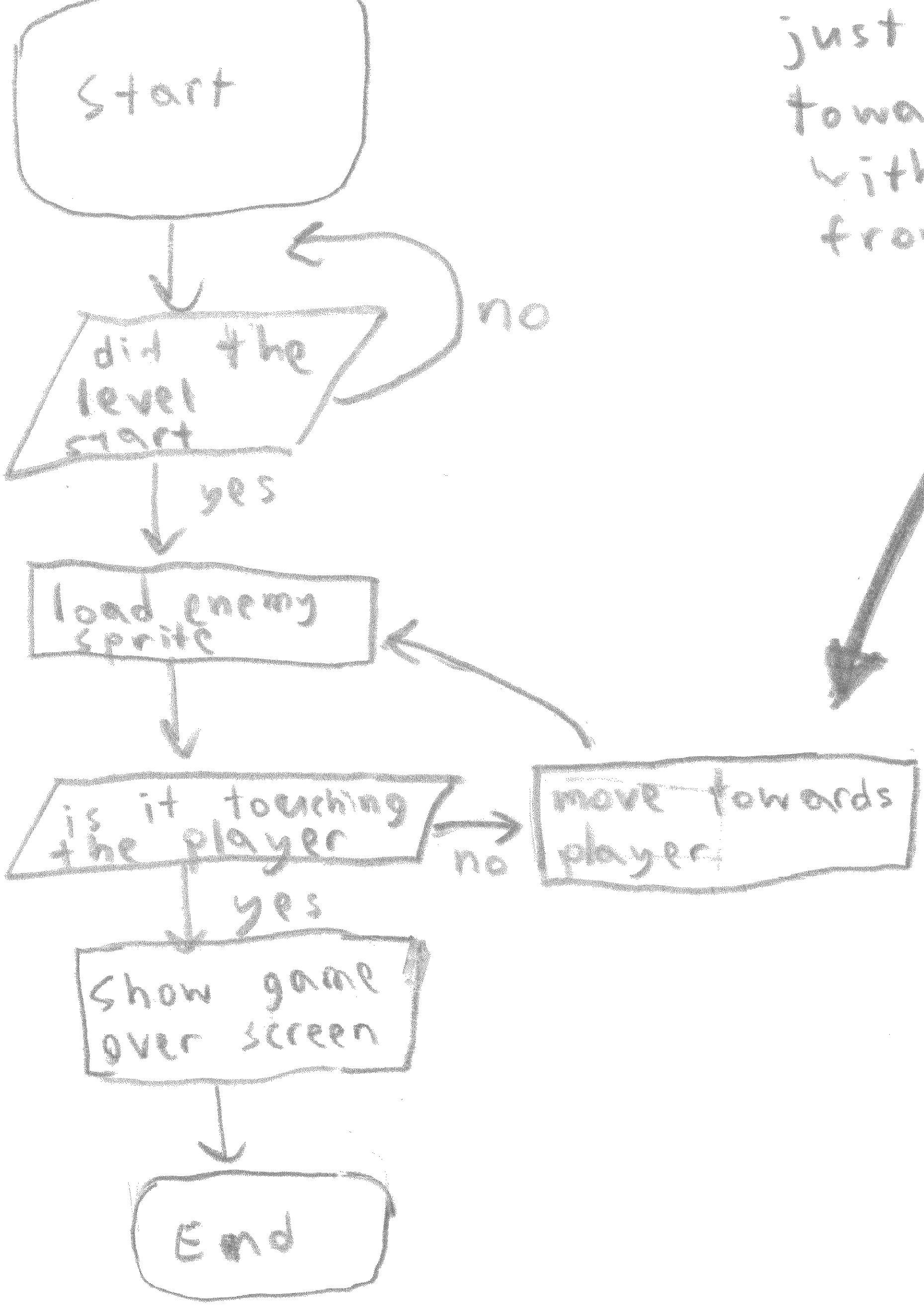
level 2 & 3 are the same but it loads in pushable boxes and enemies respectively


start
choose to play level one
load the player sprite
is the player colliding with a wall
no
yes
stop the player from moving
did the player reach the end
no
yes
end

player Flowchar



enemy flowchart



for the pushable box, just replace "move towards the player" with "move away from the player"