New Mexico Military Institute
Department of Computer Science

MCIS 1814 Fall 2023 Documentation of Final Project: Crossy Cadet

William Rosser
Amelia Hull
Isaac Armenta
Arina Gancicova
Supervisor: Afolabi, Oluwagboyega

November 21, 2023

Declaration

We, William Rosser, Amelia Hull, Isaac Armenta, Arina Gancicova, of the Department of Computer Science, New Mexico Military Institute, confirm that this is our own work and figures, code snippets, artworks, and illustrations in this report are original and have not been taken from any other person's work, except where the works of others have been explicitly acknowledged, quoted, and referenced. I understand that if failing to do so will be considered a case of plagiarism. Plagiarism is a form of academic misconduct and will be penalised accordingly.

I give consent to a copy of my report being shared with future students as an exemplar.

We give consent for my work to be made available more widely to members of NMMI and public with interest in teaching, learning and research.

William Rosser, Amelia Hull, Isaac Armenta, Arina Gancicova
November 21, 2023

Terms and Definitions

- IDE: IDE stands for integrated development environment. Programmers use it to code more easily.
- Class: Classes are used to create user-defined objects that contain groups of related variables and functions.
- Object: A value that is automatically created by the interpreter when executing a line of code.
- Attribute: The object a set of attributes that determines the data and behavior of the class.
- Method: A method is a function defined within a class.
- Instance: Creates an instance, which is an individual object of the given class.
- Class Interface: Consists of the methods that a programmer calls to create, modify, or access a class instance.
- Package: A directory that, when imported, gives access to all of the modules stored in the directory.
- File: Information or data which stays in the computer storage devices. Python allows you to manipulate these files and also the file system contains the files and directories.
- Module: A file that contains code that can be used by other modules or scripts.
- Rootfile: Within python, the rootfile format is used to store all HEP data. The rootfile contains the files and directories and critical for systems operation, including the device directory and programs for booting the system.
- Loop: A program that repeatedly executes the body while the expression is true. As soon as it is false, it terminates. For Loop: loops over each element in a container one at a time, assigning a variable with the next element that can be used in the body. While Loop: construct that executes an indented block of code as long as the expression is true.
- Function, function parameters, return Function: A function is a list of statements that can be executed simply by referring to the function's name. Function Parameters: input specified in a function definition. Return: use inside a function/method to send results back to the caller.
- String: Input obtained by input() is any text that a user typed, including numbers, letters, or special character such as # or @. Such text in a computer program is called a string.

- Lists: A container created by enclosing variables/literals with brackets [].

  Definitions taken from the zyBook textbook (Programming in Python 3) by Bailey Miller and others. Miller, Bailey. Programming in Python 3. Zyante Inc, 2023.

Chapter 1

Introduction

1.1 Background

The context of our project is we wanted to create a game in order to demonstrate our competency in programming, and be able to criticize the disciplinary system of NMMI while doing so. We see that life at NMMI is analogous to a game of crossy road, where there is constant threat of having ones progress being hindered by TLAs with nothing better to do than harrass cadets. As a consequence of our paradigm, we decided to create a fun visual representation of it.

1.2 Aims and objectives

Aims: Our aim is to create a multilevel crossy road type game that has cutscenes, dialogue, and visuals that are themed after NMMI in order to demonstrate the hardship that cadets have to go through.

Objectives: Our objectives are:

- to create sprites for our NMMI themed visuals
- to collect mp4 audio that is NMMI themed in order to immerse the user in the NMMI environment
- have game mechanics that function correctly such as movement, getting caught, and beating a level.

1.3 Solution approach

Our solution to reach our aims and objectives was to use references and creating our NMMI version of what we saw. The Methodology chapter contains our code with an explanation of how the mechanics work and the backend development not seen while playing the game.

1.3.1 Root File

The root file is where all the packages, programs, and PNGs are located. If any of it is located elsewhere it will not be processed when running the game.

1.3.2 Sprites

In order to create sprites we used the online website pixil art, and then exported our sprites as PNGs and imported them into our main code through the root file.

1.3.3 Audio

We want to use NMMI themed audio in order to immerse our player in the NMMI world. We felt the most relevant song was *The Old Post*.

1.3.4 Game Mechanics

We need to be able to create movement mechanics and branches that initiate if the user does something. We do so through the use of the pygame module, which contains functions, classes, and methods that aid us in creating the mechanics we need for our game.

1.4 Summary of contributions and achievements

We have created a multilevel, immersive, NMMI themed game that demonstrates the daily struggle a cadet faces.

1.5 Organization of the report

This report is organized into five chapters. The second chapter is the literature review, detailing the sources we used to guide our project. The third chapter is the methodology, which details our source code accompanied by annotations explaining it. The fourth chapter is the detailed Timeline of our project. The fifth chapter is the conclusion, reflection, and analysis of our project. Additionally, as you saw, an appendix has been included at the beginning of the report detailing terms and their definitions important for the readers comprehension of the report.

Chapter 2

Reference Review

The two references we used were as follows:
https://www.pygame.org/docs/tut/PygameIntro.html https://www.youtube.com/playlist?list=PL672HbqQIxzvXikVwFcQfd3xb2pwCL8zM

2.1 Pygame Documentation

We used the Pygame's official documentation articles in order to understand their modules in order that we may employ them correctly for our own use into our code. Preface: The code may appear different in the methodology due to some short hands, but it will be obvious to the audience that a function in the methodology will be one of the ones detailed in the literature section. Further, any arguments are not provided in the reference review for the sake of not being repetitive. the following lines of code will be annotated with its purpose underneath the code. The relevant sections we analyzed were:

1. The pygame display module.

   This module offers control over the pygame display. Pygame has a single display Surface that is either contained in a window or runs full screen. Once you create the display you treat it as a regular Surface.
   code:pygame.display.set_caption
   purpose: Set the current window caption;If the display has a window title, this function will change the name on the window.

   code: pygame.display.set_mode
   purpose: Initialize a window or screen for display

   code: pygame.display.update
   purpose: Update portions of the screen for software displays.

2. We also used the pygame sprite module, which is a pygame module used to create basic game object classes. The base class for visible game objects. Derived classes will want to override the Sprite.update() and assign a Sprite.image and Sprite.rect attributes. The initializer can accept any number of Group instances to be added to.

   code:pygame.sprite.Sprite
   purpose: Simple base class for visible game objects, subclasses will be created off of this.

3. We used the pygame image module for image transfer, in order to put our NMMI themed images into the game. Load an image from a file source. You can pass either a filename, or a Python file-like object.

   code: pygame.image.load
   purpose: load new image from a file (or file-like object).

   Pygame will automatically determine the image type (e.g., GIF or bitmap) and create a new Surface object from the data.

4. We used the pygame event module to interact with events and queues. Pygame handles all its event messaging through an event queue. The routines in this module help you manage that event queue. The input queue is heavily dependent on the pygame.displaypygame module to control the display window and screen module. If the display has not been initialized and a video mode not set, the event queue may not work properly.

   code:pygame.event.get
   purpose: get events from the queue

5. We used the pygame transform module in order to transform surfaces. Surface transform is an operation that moves or resizes the pixels. All these functions take a Surface to operate on and return a new Surface with the results.

   code: pygame.transform.scale
   purpose: resize to new resolution Resizes the Surface to a new size, given as (width, height).

   An optional destination surface can be used, rather than have it create a new one. This is quicker if you want to repeatedly scale something. However the destination must be the same size as the size (width, height) passed in. Also the destination surface must be the same format.

6. We used the pygame key module, which is used to enable the user to do work with their keyboard. Returns a sequence of boolean values representing the state of every key on the keyboard. Use the key constant values to index the array. A True value means that the button is pressed.

   code: pygame.key.get_pressed
   purpose: get the state of all keyboard buttons.

7. We used the mask module, which is used for image masks. A Mask object is used to represent a 2D bitmask. Each bit in the mask represents a pixel. 1 is used to indicate a set bit and 0 is used to indicate an unset bit. Set bits in a mask can be used to detect collisions with other masks and their set bits.

   code: pygame.mask.from_surface
   purpose: Creates a Mask from the given surface

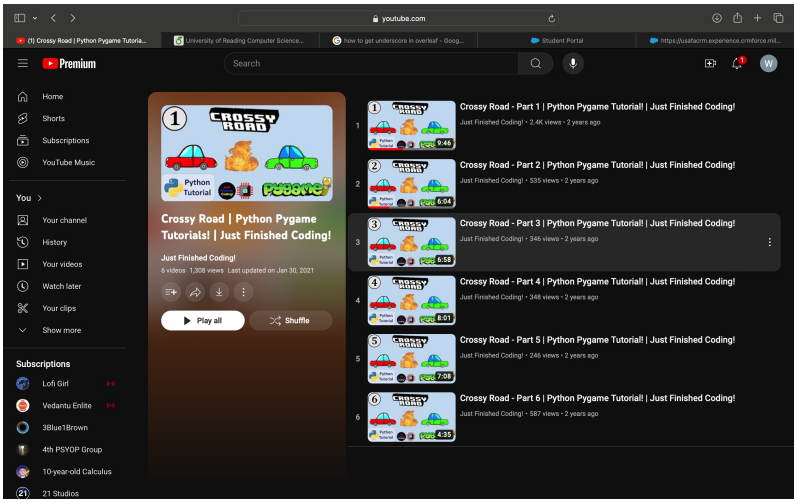
   We used the pygame font module, which is used for loading and rendering fonts. Return a new Font object that is loaded from the system fonts. The font will match the requested bold and italic flags. Pygame uses a small set of common font aliases. If the specific font you ask for is not available, a reasonable alternative may be used. If

a suitable system font is not found this will fall back on loading the default pygame font.

code: pygame.font.SysFont
purpose: create a Font object from the system fonts

2.2 YouTube Playlist Guidance

The YouTube playlist helped guide us on the code we needed to create in order for us to create certain characteristics for our game. Below is a picture of the YouTube Playlist we used.



2.3 Summary

In this chapter, we discussed the two sources we used to help create our project. We used the pygame official documentation modules in order to create our game and guide us on what methods to use to achieve our goals, and we used the YouTube playlist in order to have a general idea of what our objectives should be in order to complete the project.

Chapter 3

Methodology

We mentioned in Chapter 1 that we wanted our project to simulate NMMI and the constant threat of being hindered by TLAs. We created our game and have provided the code along with explanations to help understand our work.

First is importing the 'pygame' module because most of the functions used will come from that. We also changed it to be able to called by the letter "p", this is to save time when writing the code. 'time' is also imported and that is used to lag time in our explosions.

import pygame as p

import time

creating a character: this is our cadet. The values of the PNG are set with the width and height. The 'self.vel' is the speed of the cadet during the game. The x and y values are where the cadet starts at the begining of the game.

class Cadet(p.sprite.Sprite):

def__init__(self):

super().__init__()

self.x = 50

self.y = HEIGHT / 2

self.vel = 4

self.width = 64

self.height = 64

Here images are loaded and scaled to have the correct dimensions for movement and collision.

self.cadet1 = p.image.load('cadet1.png')

self.cadet2 = p.image.load('cadet2.png')

self.cadet1 = p.transform.scale(self.cadet1, (self.width, self.height))

self.cadet2 = p.transform.scale(self.cadet2, (self.width, self.height))

self.image = self.cadet1

self.rect = self.image.get_rect()

self.mask = p.mask.from_surface(self.image)

**This update is updating the screen to make sure the player sees movement and collision properly.

def update(self):

self.movement()

self.correction()

self.checkCollision()

self.rect.center = (self.x, self.y)

This is the movement for the cadet/player. Values for Left and Up are negative because the position of the pixels start from the top left so as the character moves closer to the top, the values would decrease. The opposite for both Right and Down as values will increase going to the bottom right.

def movement(self):

keys = p.key.get_pressed()

if keys[p.K_LEFT]:

self.x -= self.vel

self.image = self.cadet2

elif keys[p.K_RIGHT]:

self.x += self.vel

self.image = self.cadet1

if keys[p.K_UP]:

self.y -= self.vel

elif keys[p.K_DOWN]:

self.y += self.vel

This correction is to set the borders of the game for the sprites, aka the characters. Without correction the character would go off screen.

def correction(self):

if self.x - self.width / 2 ¡ 0:

self.x = self.width / 2

elif self.x + self.width / 2 ¿ WIDTH:

self.x = WIDTH - self.width / 2

if self.y - self.height / 2 ¡ 0:

self.y = self.height / 2

elif self.y + self.height / 2 ¿ HEIGHT:

self.y = HEIGHT - self.height / 2

This is checking for collision between the cadet and the car_group. If true, then the explosion function will occur.

def checkCollision(self):

car_check = p.sprite.spritecollide(self, car_group, False, p.sprite.collide_mask)

if car_check:

explosion.explode(self.x, self.y)

There are two sprites slow and fast; slow is the TLA; fast is the Golf cart. Each one has a different x value because they are located at different points in the game. The speed is also different for each one to add some difficulty to the game.

class Car(p.sprite.Sprite):

def__init__(self, number):

super().__init__()

if number == 1:

self.x = 190

self.image = p.image.load('Slow Car.png')

self.vel = -4

else:

self.x = 460

self.image = p.image.load('Fast Car.png')

self.vel = 6

self.y = HEIGHT / 2

self.width = 80

self.height = 80

self.image = p.transform.scale(self.image, (self.width, self.height))

self.rect = self.image.get_rect()

self.mask = p.mask.from_surface(self.image)

def update(self):

self.movement()

self.rect.center = (self.x, self.y)

The image and update functions are similar to the cadet, but the movement is different because there are no keys and the value of the velocity is multiplied by -1 everytime that the sprite collides with the border of the window.

def movement(self):

self.y += self.vel

if self.y - self.height / 2 ¡ 0:

self.y = self.height / 2

self.vel *= -1

elif self.y + self.height / 2 ¿ HEIGHT:

self.y = HEIGHT - self.height / 2

self.vel *= -1

Here we are creating the background of the screen. There are three that are used, 'Scene' which is the barracks, 'You Win' which is the escaping PNG, and 'You Lose' which is the commadant's office.

class Screen(p.sprite.Sprite):

```
def__init__(self):
super().__init__()
self.img1 = p.image.load('Scene.png')
self.img2 = p.image.load('You Win.png')
self.img3 = p.image.load('You lose.png')
self.img1 = p.transform.scale(self.img1, (WIDTH, HEIGHT))
self.img2 = p.transform.scale(self.img2, (WIDTH, HEIGHT))
self.img3 = p.transform.scale(self.img3, (WIDTH, HEIGHT))
self.image = self.img1
self.x = 0
self.y = 0
self.rect = self.image.get_rect()
```

This rect command is what allows to configure the image and switch it to other screens. In this case it says 'topleft' because that is where the PNG will start, which is (0,0) on the grid

```
def update(self):
self.rect.topleft = (self.x, self.y)
```

This is the class for the flags. The functions are similar regarding image and updating. The difference is that collision does not change the screen but instead adds to the score or goes to the winning screen if the score is at the max.

```
class Flag(p.sprite.Sprite):
def__init__(self, number):
super().__init__()
self.number = number
if self.number == 1:
self.image = p.image.load('green flag.png')
self.visible = False
self.x = 50
else:
self.image = p.image.load('white flag.png')
self.visible = True
self.x = 580
self.y = HEIGHT / 2
self.image = p.transform.scale2x(self.image)
self.rect = self.image.get_rect()
self.mask = p.mask.from_surface(self.image)
def update(self):
if self.visible:
```

self.collision()

self.rect.center = (self.x, self.y)

def collision(self):

global SCORE, cadet

flag_hit = p.sprite.spritecollide(self, cadet_group, False, p.sprite.collide_mask)

if flag_hit:

self.visible = False

if self.number == 1:

white_flag.visible = True

if SCORE ¡ 5:

SwitchLevel()

else:

cadet_group.empty()

DeleteOtherItems()

EndScreen(1)

else:

green_flag.visible = True

Creating explosion: This goes through 8 PNGs of explosions to make a short animation. This animation is made using the costume, where it will progressively go through each image. This only occurs when there is collision between the car_group and the cadet.

class Explosion(object):

def__init__(self):

self.costume = 1

self.width = 140

self.height = 140

self.image = p.image.load('explosion' + str(self.costume) + '.png')

self.image = p.transform.scale(self.image, (self.width, self.height))

def explode(self, x, y):

x = x - self.width / 2

y = y - self.height / 2

DeleteCadet()

while self.costume ¡ 9:

self.image = p.image.load('explosion' + str(self.costume) + '.png')

self.image = p.transform.scale(self.image, (self.width, self.height))

win.blit(self.image, (x, y))

p.display.update()

self.costume += 1

time.sleep(0.1)

DeleteOtherItems()

EndScreen(0)

This is setting the score display while the game is on. The display is at the bottom of the screen in black, both of these values could be changed, along with what the message displays.

def ScoreDisplay():

global gameOn

if gameOn:

score_text = score_font.render(str(SCORE) + ' / 5', True, (0, 0, 0))

win.blit(score_text, (210, 390))

This is checking whether the flag should be hidden or shown.

def checkFlags():

for flag in flags:

if not flag.visible:

flag.kill()

else:

if not flag.alive():

flag_group.add(flag)

This makes the cars go faster after each level. This could be changed to make it easier or harder. It also adds one point to the score to make sure that the game is not going indefinitely.

def SwitchLevel():

global SCORE

if slow_car.vel ¡ 0:

slow_car.vel -= 1

else:

slow_car.vel += 1

if fast_car.vel ¡ 0:

fast_car.vel -= 1

else:

fast_car.vel += 1

SCORE += 1

This function kills the cadet, or makes it disappear from the screen. This is used in the explosion function to make it look like the cadet died.

def DeleteCadet():

global cadet

cadet.kill()

screen_group.draw(win)

car_group.draw(win)

flag_group.draw(win)

screen_group.update()

car_group.update()

flag_group.update()

p.display.update()

Like the 'DeleteCadet' this makes items disappear from the screen at appropriate times like when the screen changes to either a lose or win.

def DeleteOtherItems():

car_group.empty()

flag_group.empty()

flags.clear()

**This function turns the game off, and then changes the background, bg, to either the win or lose.

def EndScreen(n):

global gameOn

gameOn = False

if n == 0:

bg.image = bg.img3

elif n == 1:

bg.image = bg.img2

**Starting here is where the game actually runs because everything else where classes and conditions for the game. Width and Height set the border for what is visible (aka the window).

WIDTH = 640

HEIGHT = 480

p.mixer.pre_init(44100,16,2,4096)

p.init()

'win' is short for window. It is the display shown, and it is at a set value which is why the game cannot be maximized.

win = p.display.set_mode((WIDTH, HEIGHT))

Sets the caption at the top, not in the game but on the border of the window

p.display.set_caption('Crossy Cadet')

clock = p.time.Clock()

This short section is to add background to the music. It will loop continuously at the set volume

p.mixer.music.load("Old Post.mp3")

p.mixer.music.set_volume(.75)

p.mixer.music.play(-1)

This is the starting score, along with the font, size, and bold of the display.

SCORE = 0

score_font = p.font.SysFont('comicsans', 80, True)

bg = Screen()

This is setting a bunch of groups and values that can be used by the previously made functions in order for the game to run.

screen_group = p.sprite.Group()

screen_group.add(bg)

cadet = Cadet()

cadet_group = p.sprite.Group()

cadet_group.add(cadet)

slow_car = Car(1)

fast_car = Car(2)

car_group = p.sprite.Group()

car_group.add(slow_car, fast_car)

green_flag = Flag(1)

white_flag = Flag(2)

flag_group = p.sprite.Group()

flag_group.add(green_flag, white_flag)

flags = [green_flag, white_flag]

explosion = Explosion()

From here is the game actually running and utilizing all the functions made. 'clock.tick(60)' is the frames that the game runs at, if changed then the speed of the game would also change accordingly.

gameOn = True

run = True

while run:

clock.tick(60)

for event in p.event.get():

if event.type == p.QUIT:

run = False

screen_group.draw(win)

ScoreDisplay()

checkFlags()

car_group.draw(win)

cadet_group.draw(win)

flag_group.draw(win)

car_group.update()

```
cadet_group.update()
flag_group.update()
screen_group.update()
p.display.update()
p.quit()
```

Chapter 4

Conclusion

4.1 Reflection

When we initially started to create the code for the game, we had to establish what platform we would be developing our code on. We selectively picked PyCharm as our coding platform not only because it would be easy to use but because it was extremely accessible. The program also helped us analyze our code and allowed us to create a space where all of us could work simultaneously on the code. As a group, we developed different sections of the overall code. One of the largest learning experiences that took place was figuring out how to move all the different characters and how to display the images of the characters. Throughout this entire learning process, we all had to work together while also spending our individual time on different tasks within the game. While coding, we brainstormed and created different characteristics for the game. An individual was tasked with creating PNGs of pixelated art for the game and then was needed to coordinate with another team member to combine the images and the code so both would function simultaneously. One of the challenges that we ran into was figuring out a way to approach the game. We had to find out a way to not only code the entire game but to also create different images which would coincide with one another. We tried to imitate the game Crossy Road but altered its features and turned it into a version were a TLA at NMMI would chase around a cadet. Our overall objective was to create a game that had cutscenes, dialogue, and visuals that represented NMMI. By creating such a game, we were able to demonstrate not only our team work but also our competency within coding.

In retrospect, as a group we realized our game could be more difficult, and could have more levels. If we decide to expand on this project, those will be the first changes we make.

4.2 Timeline

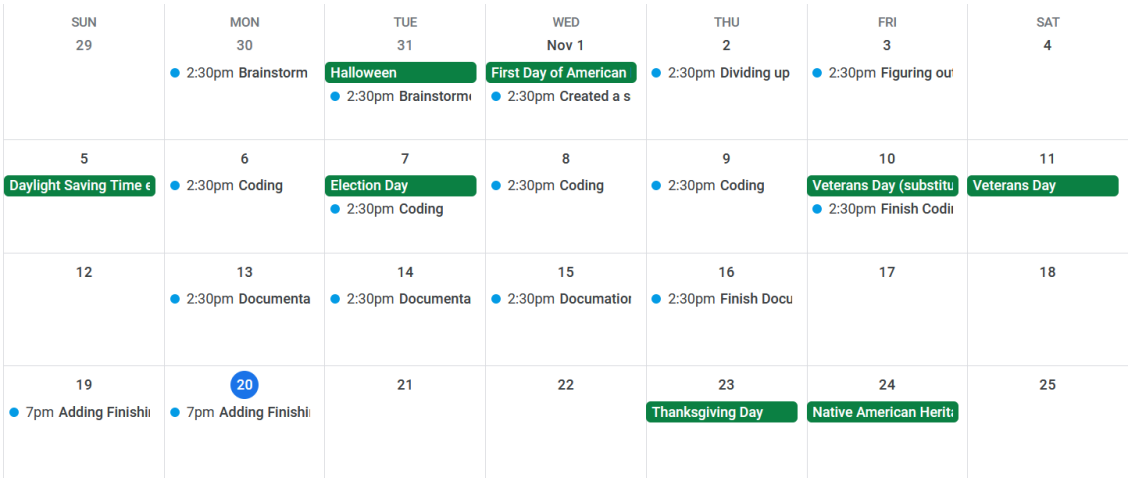


Figure 4.1: Timeline