Department of Computer Science
Technical University of Cluj-Napoca

Artificial Intelligence

Laboratory activity

Name: Dobre Mircea & Flanja Tudor Calin
Group:30434
Email: dobre.mircea2409@yahoo.com & tudorflanja@gmail.com

Teaching Assistant: Adrian Groza
Adrian.Groza@cs.utcluj.ro

Contents

Chapter 1

A1: Introduction

The Snake game is a classic and iconic arcade game in which the player controls a growing snake that moves around a confined playing area. The objective is to eat food items that appear on the screen to make the snake longer while avoiding collisions with the walls of the playing area and, more importantly, with its own tail. As the snake grows, the game becomes increasingly challenging, testing the player's reflexes and spatial awareness. The goal is to achieve the highest score possible by collecting food items and maneuvering the snake without running into obstacles. In our project we removed the collision of the snake with the walls.

Chapter 2

A2: Developement

2.1 Track the food relative to the head

In this step we gave the snake the ability to control it's movement by getting rid of the players ability to control it using the arrow keys. We repurposed the "handle_keys" function. Then we got the head position which is gonna tell us where the head of the snake is in x and y coordinates and we compared that to the food's position. In this stage the AI is going to make sure the snake understands where its head is relative to the food and pursue it, to figure out how to get closer to it. We created a tuple variable "distance", a two-dimensional vector, that's going to tell us how far from the food in the x and y direction we are. The snake is going to pursue the food in the most direct way it possibly can, but at the risk of its own body. The average score at which the snake dies is around 20 points. The code is shown below:

```
def handle_keys(self,food_pos):
        head=self.get_head_position()
        distance=(head[0]-food_pos[0],head[1]-food_pos[1])

        if distance[0] > 0:
            self.turn(LEFT)
        if distance[0] < 0:
            self.turn(RIGHT)
        if distance[1] > 0:
            self.turn(UP)
        if distance[1] < 0:
            self.turn(DOWN)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            # elif event.type == pygame.KEYDOWN:
            #     if event.key == pygame.K_UP:
            #         self.turn(UP)
            #     elif event.key == pygame.K_DOWN:
            #         self.turn(DOWN)
            #     elif event.key == pygame.K_LEFT:
            #         self.turn(LEFT)
            #     elif event.key == pygame.K_RIGHT:
            #         self.turn(RIGHT)
```

2.2 Don't allow the snake to turn into itself

We wanted to go a step further in which the snake can't turn into it's own body. Basically, it will only make a turn if it's not going to hit itself. What we want to do is to come into the "if" statements for all of our distances and add a "check" to make sure it won't turn into itself. We want to add another condition before it turns left, right, up and down. The most common way we were dying before this implementation was not actually turning into our own body but is was not turning away when we hit our own body. The average score at which the snake dies is around 23 points. The code is shown below:

```
def handle_keys(self,food_pos):
        head=self.get_head_position()
        distance=(head[0]-food_pos[0],head[1]-food_pos[1])

if distance[0] > 0 and (head[0]-GRID_SIZE,head[1]) not in self.positions:
            self.turn(LEFT)
if distance[0] < 0 and (head[0]+GRID_SIZE,head[1]) not in self.positions:
            self.turn(RIGHT)
if distance[1] > 0 and (head[0],head[1]-GRID_SIZE) not in self.positions:
            self.turn(UP)
if distance[1] < 0 and (head[0],head[1]+GRID_SIZE) not in self.positions:
            self.turn(DOWN)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            # elif event.type == pygame.KEYDOWN:
            #     if event.key == pygame.K_UP:
            #         self.turn(UP)
            #     elif event.key == pygame.K_DOWN:
            #         self.turn(DOWN)
            #     elif event.key == pygame.K_LEFT:
            #         self.turn(LEFT)
            #     elif event.key == pygame.K_RIGHT:
            #         self.turn(RIGHT)
```

2.3 Turn away before colliding with body

In this step the snake is going to be turning out of the way when it senses that the spot it's moving into is occupied by it's body. We want to create a new loop every time we check this. We declare a variable that we will use locally, "colliding". For example, if we are heading down and the square bellow is occupied, then we will set the "colliding" variable to "DOWN". This applies also to UP, LEFT and RIGHT. Now that we know if the snake is colliding, we need it to make the decision to turn out of it. To do that we will go where we we're already controlling whether to make a turn. The snake is dying at this stage because it's not calculating whether it was a better decision to move in the other direction. The code is shown bellow:

```
def handle_keys(self,food_pos):
head=self.get_head_position()
distance=(head[0]-food_pos[0],head[1]-food_pos[1])

colliding=''
if self.direction==DOWN and (head[0],head[1]+GRID_SIZE) in self.positions:
    colliding='down'
if self.direction==UP and (head[0],head[1]-GRID_SIZE) in self.positions:
    colliding='up'
if self.direction==LEFT and (head[0]-GRID_SIZE,head[1]) in self.positions:
    colliding='left'
if self.direction==RIGHT and (head[0]+GRID_SIZE,head[1]) in self.positions:
    colliding='right'

if (distance[0]>0 or (colliding=='up' or colliding=='down')) and
(head[0]-GRID_SIZE,head[1]) not in self.positions:
    self.turn(LEFT)
if distance[0]<0 and (head[0]+GRID_SIZE,head[1]) not in self.positions:
    self.turn(RIGHT)
if (distance[1] > 0 or (colliding=='left' or colliding=='right')) and
(head[0],head[1]-GRID_SIZE) not in self.positions:
    self.turn(UP)
if distance[1] < 0 and (head[0],head[1]+GRID_SIZE) not in self.positions:
    self.turn(DOWN)
for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
            # elif event.type == pygame.KEYDOWN:
            #     if event.key == pygame.K_UP:
            #         self.turn(UP)
            #     elif event.key == pygame.K_DOWN:
            #         self.turn(DOWN)
            #     elif event.key == pygame.K_LEFT:
            #         self.turn(LEFT)
            #     elif event.key == pygame.K_RIGHT:
            #         self.turn(RIGHT)
```

2.4 Determine which available turn is better

We created 4 variables left_gap, right_gap, top_gap, bot_gap in which we will store the distance between the head of the snake and it's body, in all directions possible. We want to keep track of the closest position to the head. The code is shown below for one of the cases:

```
if(head[0],head[1]-(i*GRID_SIZE)) in self.positions:
                if top_gap<i:
                        top_gap=i
```

We also added one more condition for how the snake chooses to take a turn. For example, the snake will turn left only when all the previous conditions are passed and in addition, the left_gap is bigger than the one on the right (greater or equal, in case of the same distance, it is irrelevant). The code is shown below for each case:

```
        if (distance[0]>0 or ((colliding=='up' or colliding=='down') and
        left_gap>=right_gap)) and (head[0]-GRID_SIZE,head[1])
        not in self.positions:
            self.turn(LEFT)
        if (distance[0]<0 or ((colliding=='up' or colliding=='down') and
        right_gap>left_gap)) and (head[0]+GRID_SIZE,head[1])
        not in self.positions:
            self.turn(RIGHT)
        if (distance[1] > 0 or ((colliding=='left' or colliding=='right') and
        top_gap>=bot_gap)) and (head[0],head[1]-GRID_SIZE)
        not in self.positions:
            self.turn(UP)
        if (distance[1] < 0 or ((colliding=='left' or colliding=='right') and
        bot_gap>top_gap)) and (head[0],head[1]+GRID_SIZE)
        not in self.positions:
            self.turn(DOWN)


def handle_keys(self,food_pos):
    head=self.get_head_position()
    distance=(head[0]-food_pos[0],head[1]-food_pos[1])

    left_gap = 100
    right_gap = 100
    top_gap = 100
    bot_gap = 100

    for i in range(1, int(GRID_WIDTH)):
        if (head[0], head[1] - (i * GRID_SIZE)) in self.positions:
            if top_gap < i:
                top_gap = i
        if (head[0], head[1] + (i * GRID_SIZE)) in self.positions:
            if bot_gap < i:
                bot_gap = i
        if (head[0] - (i * GRID_SIZE), head[1]) in self.positions:
            if left_gap < i:
                left_gap = i
        if (head[0] + (i * GRID_SIZE), head[1]) in self.positions:
            if right_gap < i:
                right_gap = i

colliding=''
if self.direction==DOWN and (head[0],head[1]+GRID_SIZE) in self.positions:
        colliding='down'
if self.direction==UP and (head[0],head[1]-GRID_SIZE) in self.positions:
        colliding='up'
if self.direction==LEFT and (head[0]-GRID_SIZE,head[1]) in self.positions:
        colliding='left'
if self.direction==RIGHT and (head[0]+GRID_SIZE,head[1]) in self.positions:
        colliding='right'

if (distance[0]>0 or ((colliding=='up' or colliding=='down') and
left_gap>=right_gap)) and (head[0]-GRID_SIZE,head[1]) not in self.positions:
        self.turn(LEFT)
if (distance[0]<0 or ((colliding=='up' or colliding=='down') and
right_gap>left_gap)) and (head[0]+GRID_SIZE,head[1]) not in self.positions:
        self.turn(RIGHT)
if (distance[1] > 0 or ((colliding=='left' or colliding=='right') and
top_gap>=bot_gap)) and (head[0],head[1]-GRID_SIZE) not in self.positions:
        self.turn(UP)
if (distance[1] < 0 or ((colliding=='left' or colliding=='right') and
bot_gap>top_gap)) and (head[0],head[1]+GRID_SIZE) not in self.positions:
        self.turn(DOWN)
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            pygame.quit()
            sys.exit()
             # elif event.type == pygame.KEYDOWN:
             #     if event.key == pygame.K_UP:
```

```
#             self.turn(UP)
#         elif event.key == pygame.K_DOWN:
#             self.turn(DOWN)
#         elif event.key == pygame.K_LEFT:
#             self.turn(LEFT)
#         elif event.key == pygame.K_RIGHT:
#             self.turn(RIGHT)
```

Appendix A

Your original code

```
import pygame
import sys
import random

pygame.init()

class Snake(object):
def _init_(self):
    self.length = 1
    self.positions = [((WIDTH / 2), (HEIGHT / 2))]
    self.direction = random.choice([UP, DOWN, LEFT, RIGHT])
    self.color = black

def get_head_position(self):
    return self.positions[0]

def turn(self, point):
    if self.length > 1 and (point[0] * -1, point[1] * -1) == self.direction:
        return
    else:
        self.direction = point

def move(self):
cur = self.get_head_position()
x, y = self.direction
new = (((cur[0] + (x * GRID_SIZE)) % WIDTH), (cur[1] + (y * GRID_SIZE)) % HEIGHT)
if len(self.positions) > 2 and new in self.positions[2:]:
        self.reset()
    else:
        self.positions.insert(0, new)
        if len(self.positions) > self.length:
            self.positions.pop()

    def reset(self):
        self.length = 1
        self.positions = [((WIDTH / 2), (HEIGHT / 2))]
        self.direction = random.choice([UP, DOWN, LEFT, RIGHT])

def draw(self, surface):
    for p in self.positions:
        r = pygame.Rect((p[0], p[1]), (GRID_SIZE, GRID_SIZE))
        pygame.draw.rect(surface, self.color, r)
```

```
            pygame.draw.rect(surface, black, r, 1)

    def handle_keys(self,food_pos):
        head=self.get_head_position()
        distance=(head[0]-food_pos[0],head[1]-food_pos[1])

        left_gap = 100
        right_gap = 100
        top_gap = 100
        bot_gap = 100

        for i in range(1, int(GRID_WIDTH)):
            if (head[0], head[1] - (i * GRID_SIZE)) in self.positions:
                if top_gap < i:
                    top_gap = i
            if (head[0], head[1] + (i * GRID_SIZE)) in self.positions:
                if bot_gap < i:
                    bot_gap = i
            if (head[0] - (i * GRID_SIZE), head[1]) in self.positions:
                if left_gap < i:
                    left_gap = i
            if (head[0] + (i * GRID_SIZE), head[1]) in self.positions:
                if right_gap < i:
                    right_gap = i
    colliding=''
    if self.direction==DOWN and (head[0],head[1]+GRID_SIZE) in self.positions:
            colliding='down'
    if self.direction==UP and (head[0],head[1]-GRID_SIZE) in self.positions:
            colliding='up'
    if self.direction==LEFT and (head[0]-GRID_SIZE,head[1]) in self.positions:
            colliding='left'
    if self.direction==RIGHT and (head[0]+GRID_SIZE,head[1]) in self.positions:
            colliding='right'

    if (distance[0]>0 or ((colliding=='up' or colliding=='down') and
    left_gap>=right_gap)) and (head[0]-GRID_SIZE,head[1]) not in self.positions:
            self.turn(LEFT)
    if (distance[0]<0 or ((colliding=='up' or colliding=='down') and
    right_gap>left_gap)) and (head[0]+GRID_SIZE,head[1]) not in self.positions:
            self.turn(RIGHT)
    if (distance[1] > 0 or ((colliding=='left' or colliding=='right') and
    top_gap>=bot_gap)) and (head[0],head[1]-GRID_SIZE) not in self.positions:
            self.turn(UP)
    if (distance[1] < 0 or ((colliding=='left' or colliding=='right') and
    bot_gap>top_gap)) and (head[0],head[1]+GRID_SIZE) not in self.positions:
            self.turn(DOWN)
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
                # elif event.type == pygame.KEYDOWN:
                #     if event.key == pygame.K_UP:
                #         self.turn(UP)
                #     elif event.key == pygame.K_DOWN:
                #         self.turn(DOWN)
            #     elif event.key == pygame.K_LEFT:
            #         self.turn(LEFT)
            #     elif event.key == pygame.K_RIGHT:
            #         self.turn(RIGHT)


class Food(object):
def _init_(self):
    self.position = (0, 0)
    self.color = red
    self.randomize_position()

def randomize_position(self):
self.position = (random.randint(0, GRID_WIDTH - 1) * GRID_SIZE,
random.randint(0, GRID_HEIGHT - 1) * GRID_SIZE)

def draw(self, surface):
    r = pygame.Rect((self.position[0], self.position[1]), (GRID_SIZE, GRID_SIZE))
    pygame.draw.rect(surface, self.color, r)
    pygame.draw.rect(surface, black, r, 1)


def drawGrid(surface):
for y in range(0, int(GRID_HEIGHT)):
    for x in range(0, int(GRID_WIDTH)):
        if ((x + y) % 2) == 0:
        r = pygame.Rect((x * GRID_SIZE, y * GRID_SIZE), (GRID_SIZE, GRID_SIZE))
        pygame.draw.rect(surface, gray1, r)
        else:
        rr = pygame.Rect((x * GRID_SIZE, y * GRID_SIZE), (GRID_SIZE, GRID_SIZE))
        pygame.draw.rect(surface, gray2, rr)


WIDTH = 480
HEIGHT = 480
gray1 = (120, 120, 120)
gray2 = (170, 170, 170)
red = (200, 40, 40)
green = (20, 200, 50)
black = (0, 0, 0)
GRID_SIZE = 20
GRID_WIDTH = WIDTH / GRID_SIZE
GRID_HEIGHT = HEIGHT / GRID_SIZE

UP = (0, -1)
DOWN = (0, 1)
LEFT = (-1, 0)
RIGHT = (1, 0)
font = pygame.font.Font('freesansbold.ttf', 30)


def main():
    pygame.init()

    clock = pygame.time.Clock()
    screen = pygame.display.set_mode((WIDTH, HEIGHT), 0, 32)

    surface = pygame.Surface(screen.get_size())
    surface = surface.convert()
    drawGrid(surface)

    snake = Snake()
    food = Food()

    score = 0
    while True:
        clock.tick(15)
        snake.handle_keys(food.position)
        drawGrid(surface)
        snake.move()
        if snake.get_head_position() == food.position:
            snake.length += 1
            score += 1
            food.randomize_position()
        snake.draw(surface)
        food.draw(surface)
        screen.blit(surface, (0, 0))
        text = font.render("Score {0}".format(score), True, black)
        screen.blit(text, (5, 10))
        pygame.display.update()


main()
```