Snake

Introduction

Snake was a 70s arcade game which was revived in the late 90s when it was included on Nokia phones.

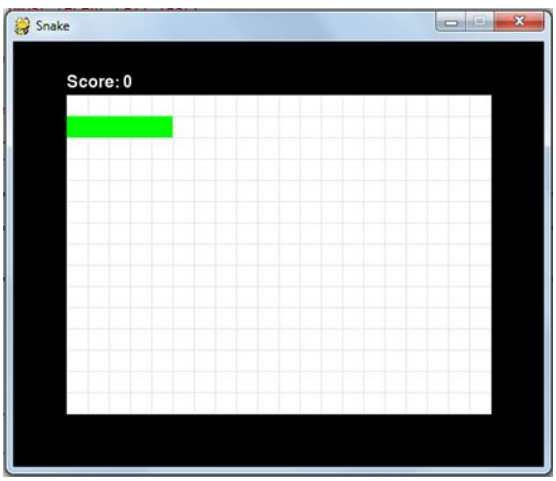
In this exercise you will create a basic Snake game in python using pygame and then customize it as you see fit.

The aim of the game is to guide the snake around the screen and eat the food that randomly appears

- Each time you eat food your snake gets longer and moves faster
- If you hit a wall or your own tail then it's game over.

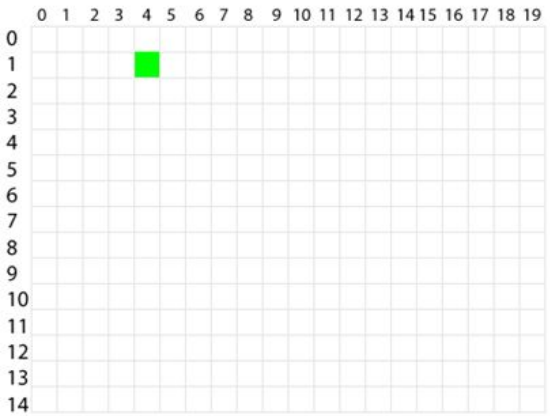
Step 0: Starting Out

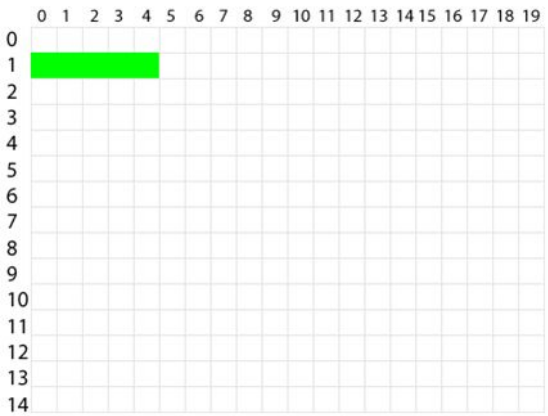
- Make sure you have the skeleton code.
- You should be able to run it and see this:
- Code to be written or modified during this exercise is marked with @
- Changing the rest is not advisable



Skeleton Code – Grid

- The play area is divided into a grid of 20 x 15 squares
- They are numbered starting from 0 (like most things in computing)
- A single cell is represented by two numbers: Horizontal first, then vertical. Beware that unlike normal coordinates the ‘y’ axis is numbered downwards
- For example the green square here is (4,1)





Skeleton Code – Snake

- The snake is represented as a list of cells
- Remember lists use [ ] and have methods like insert(), remove() etc.
- The head of the snake is the first item in the list. The tail is the last
- For example the snake above is represented as [(4,1),(3,1),(2,1),(1,1),(0,1)]

Step 1 - Movement

- The first thing to do is to make the snake move
- There is already a ‘move’ method that is called repeatedly. The only problem is it doesn’t do anything yet!
- In order to move the snake we must add one square on to the head (‘grow’), then remove one from the tail (‘shrink’)

Step 1.1 - Grow

- Find @1.1 in the code. This should be inside a method called ‘grow’.
- The list of squares that make up the snake is called ‘self.position’
- Lists have a method called ‘insert()’ to add a new square at a particular position. So to insert ‘foo’ at the front use self.position.insert(0, ‘foo’)
- The snake has an existing function called ‘self.next_space()’ which gives the square to add
- Combine these to implement ‘grow’

```
self.position.insert(0, self.next_space())
```

Step 1.2 - Shrink

- Find @1.2 in the code. This should be inside a method called ‘shrink’.
- Lists have a method called ‘pop()’ which removes the last element.
- Use this to implement ‘shrink’

```
self.position.pop()
```

Step 1.3 - Move

- Find @1.3 in the code. This should be inside a method called ‘move’
- Use the two methods you just wrote to implement ‘move’ by first growing and then shrinking the snake
- Try running the program. What happens?

```
self.grow()
self.shrink()
```

Step 2 - Controlling The Snake

- Our snake moves…. but only in a straight line
- Now to add some controls to steer it.
  - We'll use the arrow keys
- E.g. Pressing ← turns the snake towards the West
  - If already facing West does nothing
  - If facing East also do nothing – we don't want to immediately die

Step 2.1 - Change Direction

- The 'snake' class has a 'direction' property
- Indicates the direction the snake will move next
- Takes values NORTH, EAST, SOUTH, WEST
- Find @2.1 in the code. This should be inside a method called 'change_direction'
- First set the snake's direction to match the 'new_direction' parameter.
- Bonus: Only do this if the new direction doesn't send us back the way we came

```
self.direction = new_direction
```

Step 2.2 - Handle Key Presses

- Find @2.2 in the code. This should be inside a method called 'handle_events'
- This is called whenever the game detects any event such as a key press
- Pygame represents arrow keys as K_UP, K_DOWN, K_LEFT, K_RIGHT
- If the key pressed is an arrow key then call the snake's change_direction method

```
if event.key == K_UP:
    snake.change_direction(NORTH)
elif event.key == K_DOWN:
    snake.change_direction(SOUTH)
...
```

Step 2.3 - Provide Instructions

- Now the player can control the snake but we need to tell them how
- Find @2.3 in the code. This should be inside a method called ‘draw_splash_screen’
- Use the ‘draw_text’ function to provide some instructions.
    - Text is centered automatically
    - ‘y’ paramter determines vertical position

```
draw_text('Use the arrow keys to control the snake',
y=200)
```

Test the game

- You should be able to move the snake around the screen
- What happens if it reaches the edge?
- Can you do 180 degree turns?
    - If so, go back and revisit @2.1

Step 3 - Crashing

- Now to add a way to lose!
- The ‘grid’ class represents the playing area
- If the snake leaves the playing area you lose
- If the snake crashes into itself then you lose
- The task is to detect when either of these events occur

Step 3.1 - Leaving the Grid

- Find @3.1 in the code. This should be within a method called ‘within_grid’
- The grid has properties self.width and self.height
- Make the method return ‘True’ if the given cell is within the grid
- Otherwise return ‘False’

```
return x >=0 and x < self.width and
       y >= 0 and y < self.height
```

Step 3.2 - Detecting a crash

- Find @3.2 in the code. This should be within a method called ‘crashed’
- This function is called for every move
- Returns True if snake has crashed or False otherwise
- Use grid.within_grid() to test crashes into walls
- Use the ‘in’ keyword to test if the cell is part of the snake

```
if grid.within_grid(next_space) and next_space not in
snake:
    return False
else:
    return True
```

Test the game

- Try crashing into a wall
- Try crashing into the snake (may be tricky!)
- The snake is short so may have to wait for the next part to test this

Step 4 - Food and Scoring

- Now we can move the snake around the grid
- Need to add an objective
- Make food spawn on a random grid square
- When the snake reaches the food make it longer and faster
- Respawn the food somewhere else
- Increase the score by one

Step 4.1a - Pick a Random Square

- Find @4.1 in the code. This should be within a method called ‘create_food’
- First pick a random square in the grid
- Grid squares represented as pair of numbers (x,y)
- x is horizontal position and must be between 0 and 19
- y is vertical position and must be between 0 and 14
- Use random.randint(a,b) to generate a random number between a and b
- Do this twice to find a pair of numbers that form a valid grid square

Step 4.1b - Set the Square’s Contents

- Continue within @4.1 in the code. This should be within a method called ‘create_food’
- Should have a pair of random numbers
- Grid has a method set_cell(cell, contents)
  - ‘cell’ is a pair of numbers, e.g. (5,8)
  - ‘contents’ must be either self.EMPTY or self.FOOD
- Use this method to set the random square to contain food

```
x = random.randint(0, 19)
y = random.randint(0, 14)
self.set_cell((x,y), self.FOOD)
```

Step 4.2 - Draw the Food

- Look for @4.2 in the code. This should be within a method called ‘draw_grid’
- This one might be a bit tricky!
- Aim is to check each cell in the grid to see if the contents is FOOD.
    - If so then draw it using self.fill_cell((x,y), self.BLUE)
- To check one cell use get_cell((0,0)) == self.FOOD
- Need to check all of them
- How to call get_cell lots of times?
- Start by checking all values of x between 0 and 19

```
for x in range(self.width):
```

- For each value of x check all values of y between 0 and 14

```
for x in range(self.width):
    for y in range(self.height):
```

- Now we have our cell (x,y) check if it contains food

```
for y in range(self.height):
    for x in range (self.width):
        if self.get_cell((x,y)) == self.FOOD:
            self.fill_cell((x,y), BLUE)
```

Step 4.3a - Eat the Food

- Look for @4.3 in the code. This should be within a method called ‘move_snake’
- This function has calculated the space the snake will move to.
- Aim is to check whether that space contains food and if so:
    - Remove the food from that square
    - Generate more food in another square
    - Make the snake longer
    - Make the snake faster
- Look for methods in the grid and snake classes to do each. Can do one at a time and try them out.

```
if grid.get_cell(next_space) == grid.FOOD:
    grid.clear_cell(next_space)
    grid.create_food()
    snake.grow()
    snake.change_speed(-5)
```

Step 4.4 - Update the Score

- Find @4.4 in the code. This should be within a method called ‘current_score’
- Return the length of the snake minus its starting length (so score starts at zero)
- (The snake class has a method to find the length)

```
return snake.length() - 5
```

Step 5 - Top Score

- The final task is to keep track of the highest score
- Use a variable to record the highest score
- Display it at the top of the screen
- Update it at the end of the game if the record was beaten

Step 5.1 & 5.2 - Create a Global Variable

- Find @5.1 in the code.
- Create a new global variable to store the high score

```
top_score = 0
```

- Find @5.2 in the code
- Create functions to get and set the value
- Use ‘global’ keyword to refer to global variable

```
def get_top_score():
    global screen_state
    return screen_state

def set_top_score(new_score):
    global top_score
    top_score = new_score
```

Step 5.3 - Update the Top Score

- Find @5.3 in the code. This should be in a function called ‘move snake’
- Check if the current score is higher than the top score
- If so use the ‘set_top_score()’ function to update it

```
if current_score() > top_score:
            set_top_score(current_score())
```

Step 5.4 - Display the Top Score

- Find @5.4 in the code. This should be in a function called ‘draw_game_screen’
- There is an existing line to display the current score
- Add an equivalent line to display the top score
- Use the ‘get_top_score’ function

```
draw_text('Top Score: {}'.format(get_top_score()), x=350,
y=30, color=WHITE)
```

Step 5.5 - Tell the Player

- Find @5.5 in the code. This should be in a function called ‘draw_game_over_screen’
- If the current score is equal to the top score (which will already have been updated) then print a message to tell the player they got the high score

```
if current_score() == top_score:
        draw_text('You got the high score!',
font=large_font, y=100, color=BLUE)
```

Test the game

- Now the game is complete
- Try it out. Can you beat my high score of 45?
- Is it too easy or too hard?
- What parameters could you adjust?
- What else could be done to extend it?
-

Further Enhancements

Some ideas for further enhancements:

- Prevent food from appearing under the snake.
- Add a second snake to allow two players to compete. Use W, A, S and D for the other player.
- Add power-ups. Randomly spawn some special food which gives extra points, reduces the length or speed of the snake etc. Have it despawn after a short time.
- Combine the two ideas above with special food that makes your opponent longer or faster.
- Improve the graphics.
- Make the snake move more smoothly by moving fractions of a square.
- Add sound effects and music.