Did we get to the bugfix?

- At the end of last week, we didn't quite get round to adding in our bugfix.

- *There is a bug with the player code, that means that it's collision radius is always zero, so collisions aren't being detected correctly.*
- Adding a function definition for *get_collision_radius* should fix this.
- In **player.py**, add this function:

```
def get_collision_radius(self):
    return self.radius
```

- You can then get the slides for this week from here:

github.com/JamesAdey/zompy/blob/master/slides/cc_week6.pdf

Creative Computing. Week 6

(computing with cool stuff)

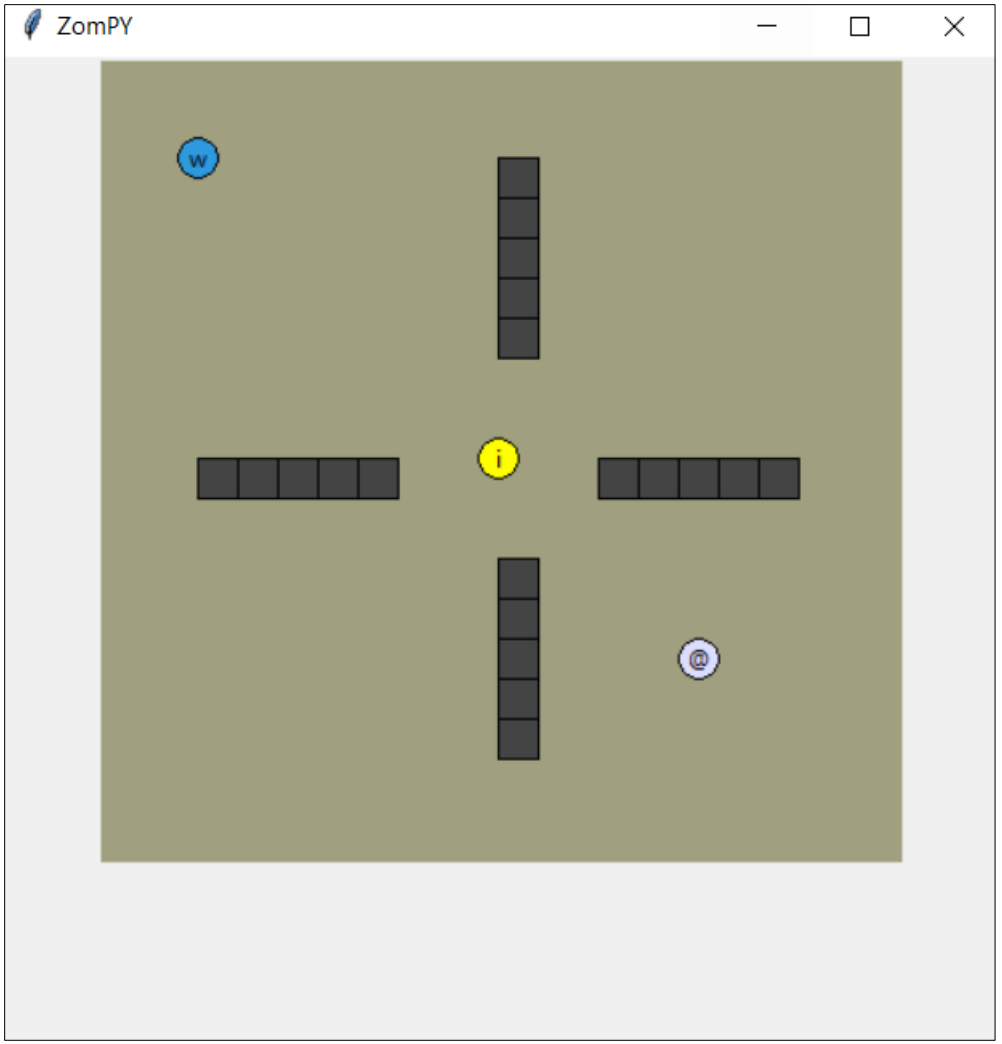
If you don't have the project yet...

- I've hosted the basic code on my GitHub

- Go to: ***github.com/jamesadey/zompy/tree/week6***
  - *This is the project containing finished code from week 5, ready for week 6.*
- Download all the files
  - *There should be a button to **download/clone***
- Open zompy_launcher.py in IDLE
  - Python version 3.x please!
- Run this file, and the game should start…

Where next?

- Our zombies can move… intelligently!
- And they can be shot…
- We have spawners…
- And item pickups!
- And random maps!



Scoring and User Interface (UI)!

- Our zombies can move… intelligently!
- And they can be shot…
- We have spawners…
- And item pickups!
- And random maps!

- Now for some UI!

Scoring & User Interface (UI).

(and zombie attacking!)

Another practical session...

- This week's session is going to be a more practical focussed.
- Part 1: Scoring
  - Adding in the map generator, and moving game setup to a level based system.
- Part 2: Health and Damage
  - Making zombies damage the player when they collide.
  - Then creating and updating a UI item to show the current player health.

Scoring is simple, UI is more complex...

- Scoring is simple, every time a zombie dies we increment a counter and use that to track how many the player has killed. We can then display this on screen using UI.

- UI in videogames is a very complex topic, getting it to scale at various resolutions, positioning all the elements correctly, then making it efficient to render as well.
- Often a lot of the rendering budget is spent on maintaining and updating UI, efficient implementations are hard to get right.
- We're going to be using ***GameObjects*** for our UI, and drawing them as we do normal game elements.

Some good practices in game UI

- **Modify the UI during the game as little as possible!**
  - Recomputing text pixels is costly, so only do it when text actually changes.
- **Re-use & recycle elements!**
  - Creating new items is costly, so where possible re-use items and just move them around instead of creating new ones.
- **Remove unnecessary code.**
  - If your UI doesn't move every frame, then leave it where it is.
  - When UI moves it will most likely cause it to be reset and have internal boundaries recalculated then undergo a full repaint of that area.
  - Areas of the canvas that don't change, don't need repainting, so lots of work can be saved by just using the old data that was there.

The Fun Part.

(actually coding stuff)

Download these slides.

github.com/JamesAdey/zompy/blob/master/slides/cc_week6.pdf

First, get these files.

- I've provided a basic game manager and a gameobject that displays text on the canvas
- 2 more files to add to the project:

  github.com/JamesAdey/zompy/blob/week6/gui_text.py

  github.com/JamesAdey/zompy/blob/week6/game_manager.py

- Then follow the instructions in the following slides.

Edit: *zompy_engine.py*

- *We need to import the **game manager** and **gui text** files*
- *Add a **gameManager** field inside **ZompyGlobals***
- *Create a **GameManager** object and link it into the engine*
- *Inside **setup_game()**, call a function to setup the user interface*
- *Create the function we just called above*

```
from navgrid import *
from game_manager import *
from gui_text import *

# these are not a class currently
# but a suite of helper methods
```

```
bulletManager = None
gameManager = None
itemManager = None
gameWorld = None
```

```
super().add_game_object(im)

gm = GameManager()
gGlobals.gameManager = gm
super().add_game_object(gm)

# create a nav grid
```

```
gGlobals.navGrid = ng

# configure the Graphical User Interface
self.setup_ui(gGlobals)

# Now decide which level to create and st
levelName = "test"
```

```
def setup_ui(self,gGlobals):
    # create a score gui and link it to the game manager
    scoreText = GUIText(x=10,y=10,baseText="Score: ")
    gGlobals.gameManager.set_score_gui(scoreText)
    scoreText.set_text("0")
    super().add_game_object(scoreText)
```

Edit: *zombie.py / zombie_gridwalker.py*

- *We can now make our zombies notify the game manager when they're killed to add score.*
- *Edit the **dead()** function in the zombie file, and add a line to notify the game manager of the zombie being killed.*
- *The **zombie_killed()** function in the game manager increases our score by 1 each time it is called.*

```
def dead(self,gameGlobals):
    # notify the game manager that we've died
    gameGlobals.gameManager.zombie_killed(self)
    # remove ourselves from the game
    gameGlobals.engine.remove_game_object(self)
```

- *You might also need to edit other zombie files too, to make those zombies give points when killed.*

What Next?

- If everything has been done right, the zombies should increase your score when you die.

- Our zombies still don't damage the player when they collide.
- We can re-use the same collision system we have for our items and detect collisions between zombies and players!
- We should also display the player's health on screen.

- The next 4 slides show how to do this…

Edit: *zombie.py / zombie_gridwalker.py*

- *Firstly, we need to import the player*

```
from gameobject import *
import player

class Zombie(GameObject):

    startX = 0
```

- *Inside the **on_start()** function, we'll register ourselves with the item manager for collision purposes*
- *Inside the **on_remove()** function, we need to unregister ourselves aswell*

```
def on_start(self,gameGlobals):
    self.startX = self.x
    self.startY = self.y
    # register ourselves with the game world as an obstacle
    gameGlobals.gameWorld.add_obstacle(self)
    # register ourselves with the gameWorld as an item
    # this means we can detect collisions
    gameGlobals.itemManager.add_item(self)

def on_remove(self, gameGlobals):
    # remove ourselves from the list of obstacles
    gameGlobals.gameWorld.remove_obstacle(self)
    # remove ourselves from the list of items
    gameGlobals.itemManager.remove_item(self)
```

- *Also add an **on_collision()** function to detect the collisions and deal the damage*

```
def get_collision_radius(self):
    return self.radius

def on_collision(self, gameGlobals, other):
    if(isinstance(other, player.Player)):
        other.take_damage(1)
```

Edit: *player.py*

- *Firstly, add* ***health*** *and* ***hurt*** *variables*
  - *We'll use the hurt variable to track when the player has taken damage so we can repaint the UI*

```
class Player(GameObject):

    health = 100
    radius = 10
    hurt = True

    colour = "#DDDDFF"
```

- *Add these lines to our* ***update()*** *function to update the ui*

```
        # draw the bullet effects
        gameGlobals.bulletManager.fire_bullet(self.x,self.y,endX,e

    # update our status in the GUI
    if(self.hurt):
        gameGlobals.gameManager.update_player_health(self.health)
        self.hurt = False

def do_inputs(self, gameGlobals):
```

- *Then add a* ***take_damage()*** *function*

```
def take_damage(self, damage):
    self.health -= damage
    self.hurt = True
```

Edit: *game_manager.py*

- *We need to add a variable to hold the new UI element*

```
zombiesKilled = 0

scoreGUI = None
healthGUI = None

def __init__(self,x=0,y=0):
    super().__init__()
```

- *Then add 2 functions to the end of the class:*

```
def set_score_gui(self, newScoreGUI):
    self.scoreGUI = newScoreGUI

def update_player_health(self,health):
    self.healthGUI.set_text(str(health))

def set_health_gui(self,newHealthGUI):
    self.healthGUI = newHealthGUI
```

- *The first function is used to update the health UI element during the game*
- *And the second function is used to initially assign the UI element at setup time*

Edit: *zompy_engine.py*

- *Finally, we just need to add another section to our **setup_ui()** function.*

```
super().add_game_object(scoreText)

# create a health gui and link it to the game manager
healthText = GUIText(x=10,y=30,baseText="Health: ")
gGlobals.gameManager.set_health_gui(healthText)
healthText.set_text("100")
super().add_game_object(healthText)
```

- *This will create a new UI text and place it underneath the score text.*

Free Play & Extensions!

- If you've got to this point, you should have a health text and a score text, **and** the health text should update when the player touches the zombies.

- What else to do?
- **Create your own levels!**
  - Go back to editing ***level_custom.py*** or make an entirely new one!
  - Can you define your own random generation functions?
- **Add obstacles that damage zombies and players**
  - Make a new obstacle, and try editing the ***on_collision()*** function to send a take damage message when an obstacle collides with a player/zombie.

Feedback

- How is it going so far?
- Is there anything you would like me to go over again?

- We have 2 weeks left, and I only want to cover 1 more topic…
  - ~~Pathfinding~~
  - ~~Levels/Maps~~
  - ~~Scoring~~
  - Special Zombies