**Cycles**

**## Program Structure**

---

The project files and folders are organized as follows:

root (project root folder)

+-- cycles (source code for game)

+-- game (specific game classes)

+-- \_\_main\_\_.py (entry point for program)

+-- README.md (general info)

Following the same pattern as in the snake game, we will have four directories:

**>Casting**

* Actor: shared logic for changing the displayed text and screen position
* Cast: stores the actors in a dictionary for later access
* Player: Player snake that keeps track of its segments
* Score: Stores the player’s score

**>Directing**

* Director: Controls the game flow and updates the actors as needed

**>Scripting**

* Action: Interface for scripts
* ControlActorsAction: Handle inputs for controlling actor behaviors
* DrawActorsAction: Draw visible actors using VideoService
* GrowPlayerTailsAction: Increase the length of the player tails each frame
* MoveActorsAction: Move the actors each frame

**>Services**

* Keyboard Service: listens to the keyboard for any inputs
* Video Service: Creates the program window and draws to the screen

**>Shared**

* Color: Hold color information
* Point: Hold positional coordinates for actor positions

**Main()**

* Creates the necessary objects and starts the game loop

**Casting**

The classes needed are the nearly same as snake, with the following changes:

**Player**

* Inherits the Actor class and is originally the code for Snake
* **\_\_init\_\_()** will include a player number attributes
* **grow\_tail()** will be changed to set the color based on the player number
* **prepare\_body()** will be changed to set the starting position and color according to the player number

**Directing**

**Director**

By utilizing polymorphism, no changes need to be made

**Scripting**

The classes needed are nearly the same as snake, with the following changes:

**ControlActorsAction**

* Add listeners for player 2’s controls (IJKL)

**HandleCollisonsAction**

* Remove handle\_food\_collisions
* Update methods to iterate through players and scores while checking for collisions

**GrowPlayerTailsAction**

* Inherits the Action interface
* execute(): is overridden to iterate through the list of players and grow their tails

**Services**

By utilizing polymorphism, no changes need to be made

**Shared**

By utilizing polymorphism, no changes need to be made

**main()**

* **Main()** will be changed to create two player objects and two scores, one for each player
* constants will be changed to start the players with only 1 segment