AI Snake

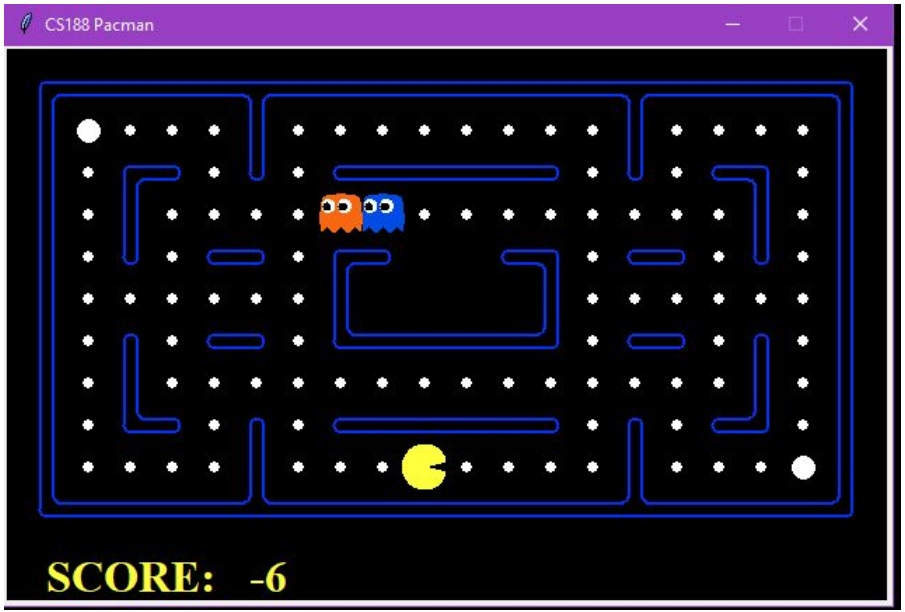
Project by Lynden Millington

What is Snake?

You've almost certainly played Snake before. The snake is the player character, and you must reach the food in order to grow and increase your score. The only obstacles are the walls and yourself, because as you grow you will become a larger obstacle.

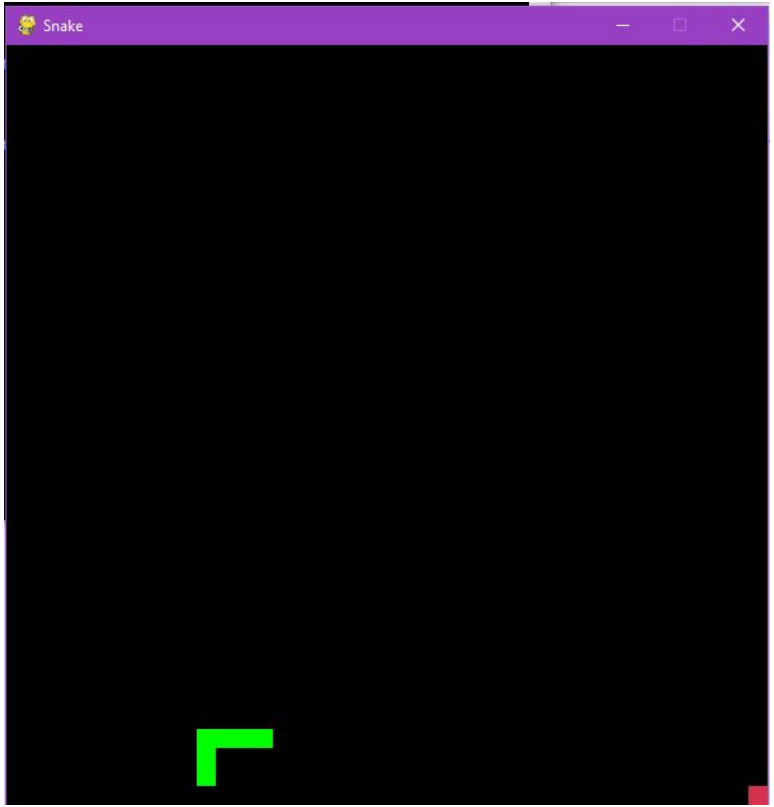
Expanding on Pacman

Snake can be compared to our Pacman projects, if we imagine that the food is a single pellet, and the snake's own body is one continuous wall (or ghost, since touching it will end the game immediately).



Using pygame to implement snake in Python

The base snake game has been implemented using features within the pygame library. The arrow keys control the snake. The graphics update after every move to reflect the changing game state.



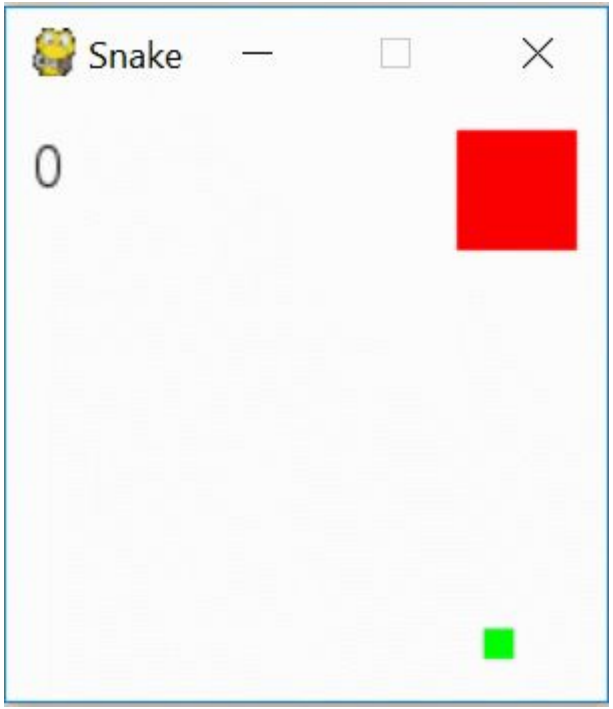
Reinforcement Learning

Snake is a very simple game. It's almost similar to our test examples for reinforcement learning in class. There's so few possible outcomes that we can use a discrete reward table, like this one. Eating food gives a positive reward, and colliding with either its own body or a wall gives a negative reward. Living also gives a slight negative reward.

Action	Reward
Eat food	1
Collision	-1
Other	-0.1

Why negative living cost?

A negative living cost is necessary in order to force the snake to learn. If we do not punish living without getting food, the snake will be less incentivized to seek out the food, and will be more likely to become trapped in a loop like this one. The snake is a length of 4, but it is continuing to double back on itself in a loop. By slightly punishing living we can avoid this dilemma.



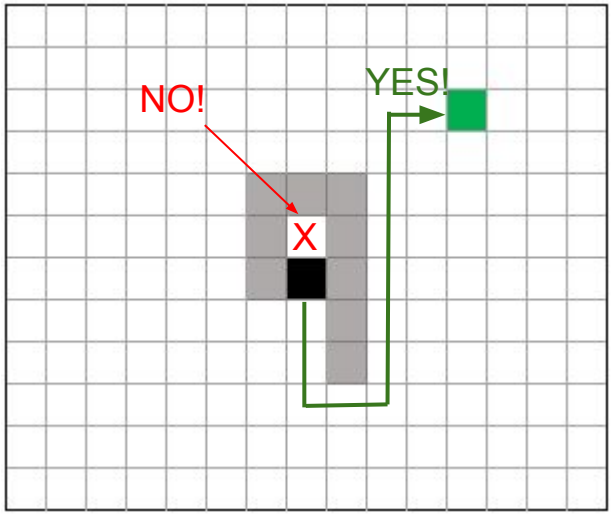
Problems with Reinforcement Learning

In this arrangement, we will eventually become pretty good at Snake! This can take a long time though, depending on the size of the board. Q-learning requires many iterations to get stable Q-values, and the possible situations are randomized every time. Not only do the head position and food position need to be taken into account, but also every combination of body positions too.

STATES	ACTIONS			
	UP	DOWN	LEFT	RIGHT
...	...	...	...	...
.... .	0.43	0.25	0.12	**0.8**
. ....	-0.5	-0.3	**0.38**	-0.15
:...:	**0.6**	0.18	0	0.53
:...:	-0.32	**0.75**	0.23	0.49
...	...	...	...	...

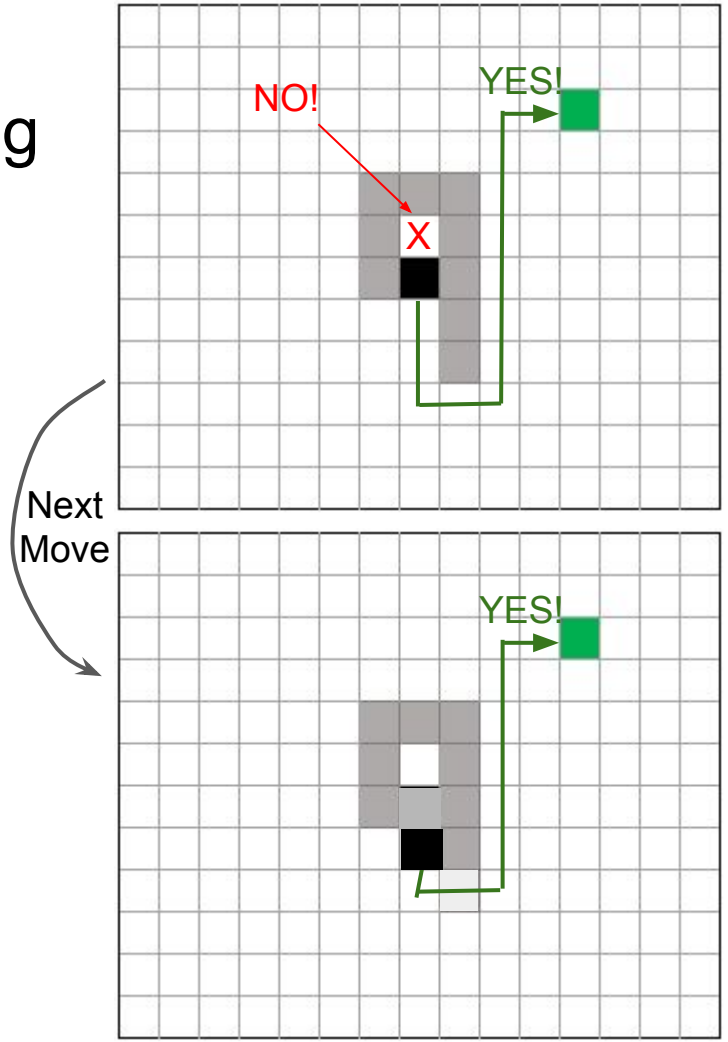
Back to basics!

Our search-based algorithms work very well in this situation. Depth- and breadth-first searches can be used to find paths to the food, and we can treat the snake's body like we treated walls in our Pacman implementations, so running into ourselves will generally be avoided from the start. I will be using the A* algorithm, since it prioritizes finding the shortest path among those it computed.



A Benefit of A* over Q-Learning

Where Q-Learning needs to develop its understanding of the state space over time, A* can calculate a new path during every iteration, so when the snake's tail recedes out of the way, we can make a new optimal path without needing to consider where every other segment of the body is at all times.



What’s Next?

The current implementation crashes here. Why? The snake has just eaten food at the head position (yellow) and a new food has spawned inside the coiled snake’s body. A* fails here, because there is no path that allows the snake to get there. But this is obviously not game over! I need a way to teach the snake to continue in some manner until it can open up this area again.