Preliminary Project Pnotes

* It’s Mario; agent will need to get as far right as possible to win levels
* BEATING LEVELS PAST 1-1 IS EXTRA BONUS CONTENT; FOCUS ON 1-1 BEFORE IMPLEMENTING THINGS LIKE PALLETTES, SUBAREAS, LAKITUS etc.
* Water levels do exist, but probably won’t be tested in basic testing; if you have time do account for them
* We only get Mario’s position, the picture of the screen (as an array of arrays containing RGB values), and a few other variables like powerups; identifying what is ahead is for the agent to do
* Main issue I can think of is distinguishing between brown blocks and brown Goombas (and similar issues)
* Identifying enemies could involve checking each pixel on the screen (besides ground ones); if you see a pixel that matches the enemiy’s top-leftmost pixel, check the surrounding pixels for the rest of the enemy
* Basic rule-based design would include ‘go right whenever possible; if Goomba, Koopa Troopa, bottomless pit etc. is in front of you jump; if your x position doesn’t change for 3 seconds jump as high as possible while going right; if it still doesn’t change go left for a second; if mushroom/fire flower is in front of you run towards it; spam run if you have fire flower, if above pipe try to go down in case there’s a subarea; same with right-facing pipes’
* Libraries are allowed, just mention them in the report
* Include a README.md file with your submission
* Online solutions exist; feel free to look at them, but DO NOT copy them
* I’m using Poetry for my machine’s installation; use poetry run python marioScript.py to run it
* Use SuperMarioBros-v0 as the environment; it has all the graphics
* Check pytorch tutorial; even if you don’t use pytorch itself its design lends itself to learning reinforcement agents
* Use ‘poetry run python ruleBasedMario.py’ to run the code
* Remember that the actions possible are:
* COMPLEX\_MOVEMENT = [
* ['NOOP'],
* ['right'],
* ['right', 'A'],
* ['right', 'B'],
* ['right', 'A', 'B'],
* ['A'],
* ['left'],
* ['left', 'A'],
* ['left', 'B'],
* ['left', 'A', 'B'],
* ['down'],
* ['up'],
* ]
* HUD COIN: 252, 160, 68 in light colouring; at wide part leftmost pixel is at index 90 of row 26-31; highest part is index 91-92 of row 25
* Screen is 256\*256 pixels, each with one triple representing their RGB value in the observation table. Each subtable of observation is a row, with the rows listed top to bottom. Each triple in a row is a pixel, with pixels taken left to right. In other words, entries are just left to right top to bottom
* For the basic ground theme like 1-1: Sky is (104, 136, 252), ground tile is (228, 92, 16) with bits of pure (0,0,0) black and (240, 208, 176) tan, ? blocks alternate between primarily ((228, 92, 16) [dark] and (252, 160, 68) [light] with unchanging bits being black and ground brown
* Idea: use 16\*16 grid based search to simplify block search (won’t work for Goombas and the like)
* Idea: categorise objects as just ‘enemy’, ‘hazard’, ‘block’ etc.
* Colours in ground theme:
* (104, 136, 252) light blue: sky
* (228, 92, 16) brown: ground tiles, bricks, ? blocks at some stages + ? edges, coins at some stages, Goombas
* (240, 208, 176) tan: Goomba stalks, brick edges,
* (252, 160, 68) yellow: coins at their brightest, ? blocks at their brightest, Mario’s face
* (0, 0, 0) black: edges of many blocks, Goomba eye[brow]s
* Colours in underground theme:
* Sky: 0,0,0
* None of the usually brown things (bricks, goombas) have any Red value

Misc. notes [DELETE BEFORE SUBMITTING]:

Our agents are Hand Implemented Rule based agent and PPO from Stable Baselines

When we make changes to existing code (Laurens, pytorch tutorial) document it!

‘poetry run nes\_py --rom super-mario-bros.nes --mode human’ for human controlled Mario

Analyse and contrast the performance of the chosen AI methods.

• Discuss their respective strengths, weaknesses, and suitability for playing Super Mario Bros.

* Rule based is easy to understand; parameters can be tweaked easily to make it jump shorter, etc.,
* Rule based cannot adapt on its own; new palettes like underground or new enemies like Lakitus require adding them to the enemy recognition program, coding new actions to avoid them, etc.
* Experiments could include testing speed of level completion (be it time, frames or actions) or the amount of memory used; many of the listed examples don’t really work with a rule-based agent that doesn’t learn
* Rule based takes 2374 steps total; 1618 to beat 1-1

You will notice that gym-super-mario-bros reward function assumes the objective of the game is to move as far right as possible. You are encouraged to come up with other performance and evaluation metrics for your agents. Novel and interesting metrics that you come up with will be rewarded.

* Points as a metric (collect coins/powerups gives lives and increases survivability; death penalty prevents infinite lives trick from causing problems
* Progression as a metric (get as far into the game as possible in terms of levels, or alternatively beat 8-4 from 1-1 as fast as possible; rewards finding the Warp Zones)

Includewhatvisualizationtechniquesyouusedtogaininsightsintotheagent’sdecision- making process.

Include what debugging/profiling tools you utilised to optimize the algorithms and enhance performance.

* Besides what was already in Lauren’s code, printing to the terminal when decisions are made helps identify actions
* Freezing the game when a decision is made (via spamming the terminal with 250,000 messages) helps to identify exactly what constitutes a scenario where said decision is made