AIST1110 Introduction to Computing Using Python

Group Project Specification

2022-23 Term 1

By Dr. King Tin Lam

Form a Group

- Work in pairs.
  - 2 members per group.
- Please fill in this Google form ASAP by Nov 16 (Wed) 23:59 to indicate your preference in grouping:

  https://forms.gle/hmYrrPe6iv3GtPNw9

Learning Outcomes

- This project can be said an integrated use of many Python programming skills:
  - Control Flow
  - File I/O
  - OOP
  - NumPy
  - Matplotlib
  - TensorFlow Keras (optionally)

Preparation Work

- Discuss with your partner on the game topic
- Design your game
  - Goal
  - State
  - Actions
- Collect or make necessary game resources
  - sprite images, backgrounds, sounds, etc.
- Organize all things under a game folder:
  - Source
  - Images
  - Sounds
  - Output files
  - ...

Project Deliverables

- Game Folder (Submitted as a zip)
  - Python source files
  - Files listing all packages installed in your virtual environment
  - Trained model file(s)
    - For Q-learning, the model file is a dump of your Q-table, typically as text file or JSON file. (Binary file is not preferred here; text allows easier checking/debugging).
    - For deep Q-learning, the model file is a snapshot of your trained network, typically in hdf5 format (file extension .h5).
    - You may provide several versions, say if you tried different problem sizes or difficulty levels of the game.
- Report
  - A Word or PowerPoint document summarizing your project

Project Report

- Format: Either Word or PowerPoint document
- Length
  - Word: within 8 pages, but we expect 5 pages only
  - PowerPoint: within 25 pages, but we expect less than 10-20 pages only
- Content
  - Genre of your game
  - How to play your game?
    - Goal of the game
    - What actions to be done by the human player?
  - How to train your agent?
  - How to test your agent?
  - UML class diagram show the design of your classes (only essential ones if too many)
  - Clear specification of game state and action
  - List and explain the command-line arguments for each program
  - Experimental data charts (produced by Matplotlib)

Project Submission

- Submit to Blackboard.
- Compress your game folder into a zip file.
- Note that Blackboard restricts that each single file uploaded should not exceed 200MB.
- So, please ensure your zip file size is within 200MB.
  - If you really have difficulty to meet this requirement, split your zip file into multiple volumes like .zip.001, .zip.002, ... Most archiving tools support this feature.

Two Parts

- Part 1: Python Game Development Using PyGame and OpenAI Gym
  - Delivers a game program that can be played by human.
  - We interact with the game GUI using keyboard and/or mouse.
- Part 2: Reinforcement Learning for Playing the Game
  - Delivers an AI agent (another program) that can play the game developed automatically.
  - We can visualize the gameplay on the GUI but not to interact or interfere it.

Part 1 (Game Development)

- Your game must be implemented based on the latest version of
  - PyGame
  - OpenAI Gym
- Your PyGame is wrapped up as a gym environment so that we can
  - play it in a human mode (in which the program listens to our keyboard/mouse events), or
  - visualize the gameplay GUI in a read-only human mode which does not handle our keyboard/mouse events, or
  - expose the game in a non-GUI mode which is more efficient for agent training

Please check this out to learn how to create your custom Gym environment.

Part 2 (Reinforcement Learning)

- We will accept either one of the following algorithms for implementing the agent:
  - **Q-Learning**: build a so-called Q-table (say in form of a numpy array) for choosing the best action for a particular game state; using purely Python coding skills; no need to use and learn any machine learning framework.
  - **Deep Q-Learning**: a model to be implemented using TensorFlow 2 from scratch.
    - Note that it must be TensorFlow to match our learning outcome, which we will cover in Week 12. So, PyTorch or other ML frameworks are NOT accepted in this project.

Part 2 (Reinforcement Learning)

- You only need to implement one training algorithm, either Q-learning or deep Q-learning, no need to implement both.
  - There exist variants of the method such as double Q-learning that generally converges faster. Try not to choose the variants for our ease of grading unless your game really needs the better variant.
- If you choose to use deep Q-learning (a.k.a., deep Q-network, DNQ), you must implement it from scratch.

Part 1

Design and Develop Your Game

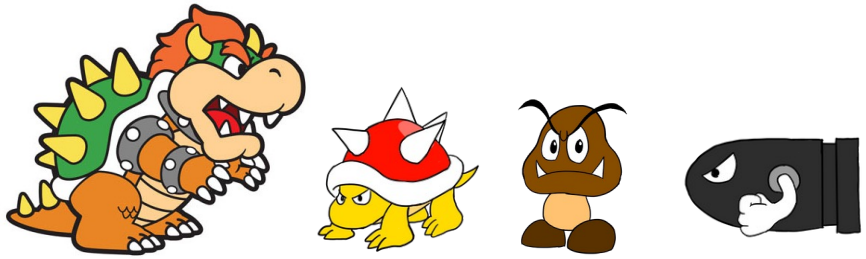
- We expect a team to design and develop the code of the whole game from scratch.
- To save time, you can search, download and reuse any resource files like sprite images, background images, audio files of sound effects and background music, font files (.TFF), etc. from the Web, with proper acknowledgments of the resource providers.

Video Game Genres

- Action
- Action-adventure
- Adventure
- Shooter
- Sports
- Race
- Platformers
- Role-playing
- Simulation
- Strategy
- Puzzles
- Cards

Basic Game Design

- Goal of the game
  - e.g., to beat enemies; to avoid obstacles, to solve a puzzle
- Movement of the player
  - e.g., up, down, left, right
- Interaction with other objects
  - e.g., colliding with enemies causes health point reduction, colliding with power-ups gets a more powerful weapon
- Game over condition
  - e.g., hit an obstacle; health points drop to zero



Implement as an Open Gym Environment

- Your game program must be subclassing `gym.Env`.
- And implementing the following methods:
  - `__init__(self, render_mode)`
  - `step(self, action)`
    - `return (observation, reward, terminated, truncated, info)`
  - `reset(self, [seed,]...)`
    - `return (observation, info)`
  - `render(self)`
  - `close(self)`

Design of Actions

- Every mouse click or key press on the keyboard should be translated into a certain action in the game.
- An action is represented by a number for ease of accessing the Q-table.
- PyGame supports joysticks but let’s skip it. We won’t test your game using a joystick controller in this project.

Design of Game State (Observation)

- The Q-learning training code seems quite standard and could be easily found on the Web.
- Perhaps the most important and challenging is the design of your game state!
- Note that you will use a game state to query or index the Q-table.
- How you implement the Q-table is also a design choice.

Design of Game State (Observation)

- For example, for the GridWorld gym env, the game state can be modelled as a tuple storing the (x, y) position of the blue ball and the (x, y) position of the red square.
  - You can use a vector (1D numpy array) to store the 4 position integers and use a numpy 2D array to implement the Q-table to store the Q-values. The table has enough rows to represent all the states; and enough columns to represent all the actions.
  - 4 actions: left, right, up, down
  - How many game states possible for a 5x5 game board?
    - 5 x 5 x 5 x 5 = 625 (ignore some states which are impossible, those with same locations for the blue ball and red square)
  - So, your Q-table has 625 rows x 4 columns.
  - Each array index represent one state.
  - How can we map a vector to the array index integer to access the corresponding row in the Q-table? You can devise your own hashing scheme or consider our encoding approach in the qtest script provided in GridWorld.zip.

Design of Game State (Observation)

- Alternatively, you can also use a dictionary to implement the Q-table, and tuples as keys to access the Q-values (represented by a 4-element Python list, say).
- For example, for a game state [1, 2, 3, 4], convert it to s = (1, 2, 3, 4) and access the Q-value via Q-table[s][action].
  - Advantage: Python dictionary will take care of the required hashing, and you don’t need to do it yourself.
  - Disadvantage: NumPy arrays are optimized and are generally faster.

Design of Game State (Observation)

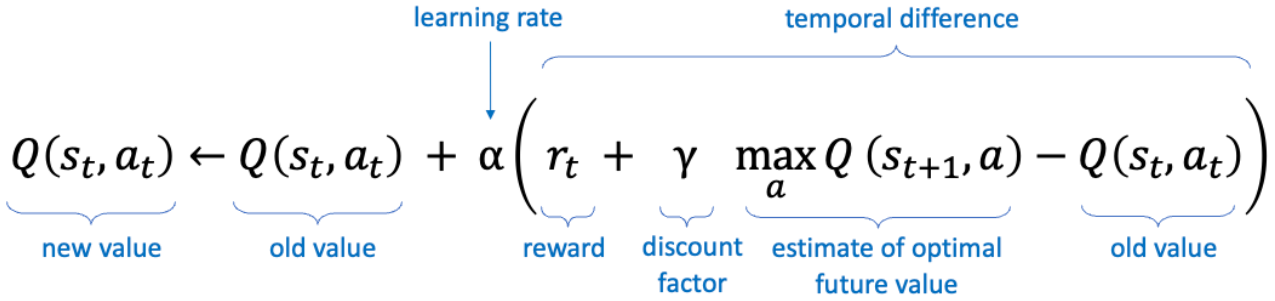
- We don’t assume having a powerful GPU for the agent training.
- So, your state must be simple enough, e.g., a vector instead of the whole game screen picture.
- Otherwise, it is hard for us to grade.
- Try to make the time of training your agent to achieve a “working” gameplay be preferably within 20 minutes. Here, “working” means suboptimal but still quite better than a random agent. Your agent may still make wrong decisions due to insufficient training episodes.
- Actually, you may train your agent for a longer time than specified above to achieve a smarter gameplay and submit your trained model file(s) to us for testing.

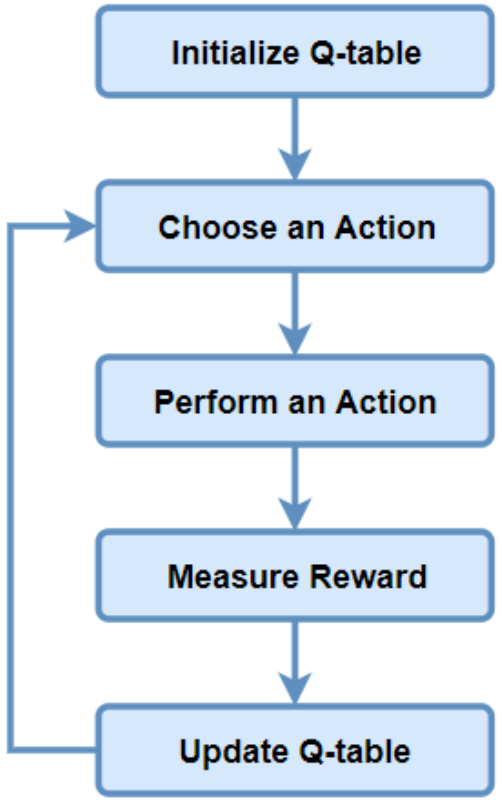
Part 2

Q-Learning

- The algorithm is not too hard.
- This article is a good reference for you to learn this technique.
- We will explain more during the tutorial.

- Basic idea is build a so-called “Q-table” for choosing the best action for a particular game state.
- This table records the quality (Q-value) of an action towards a given (game) state.
- Reinforcement learning is to use trials and errors to explore the environment. We perform random actions, measure their resulted rewards and update the corresponding Q-values in the table using the Bellman equation:





Q-learning

- Random actions can explore the state space to improve Q-values.
- But over time, we will reduce the time on doing exploration, and increase the time on exploiting the table built so far.
- This can be controlled by an ε-greedy policy:
  - Choose a constant ε between 0 and 1.
  - With probability ε, perform a random action.
  - With probability 1 – ε, perform the optimal/greedy action.
  - Instead of constant, slowly move it towards greedy policy: ε -> 0
- Pseudocode for implementation:

Algorithm 1: Epsilon-Greedy Q-Learning Algorithm

Data: α: learning rate, γ: discount factor, ϵ: a small number
Result: A Q-table containing Q(S,A) pairs defining estimated optimal policy π^*

```
/* Initialization                                                    */
```
Initialize Q(s,a) arbitrarily, except Q(terminal,.);
Q(terminal,.) ← 0;
```
/* For each step in each episode, we calculate the
   Q-value and update the Q-table                                    */
```
for *each episode* **do**
  ```
  /* Initialize state S, usually by resetting the
     environment                                                     */
  ```
  Initialize state S;
  for *each step in episode* **do**
    do
      ```
      /* Choose action A from S using epsilon-greedy
         policy derived from Q                                       */
      ```
      A ← SELECT-ACTION(Q, S, ϵ);
      Take action A, then observe reward R and next state S';
      Q(S, A) ← Q(S, A) + α [R + $\gamma \max_a$ Q(S', a) - Q(S, A)];
      S ← S';
    while *S is not terminal*;
  end
end

Deep Q-Learning

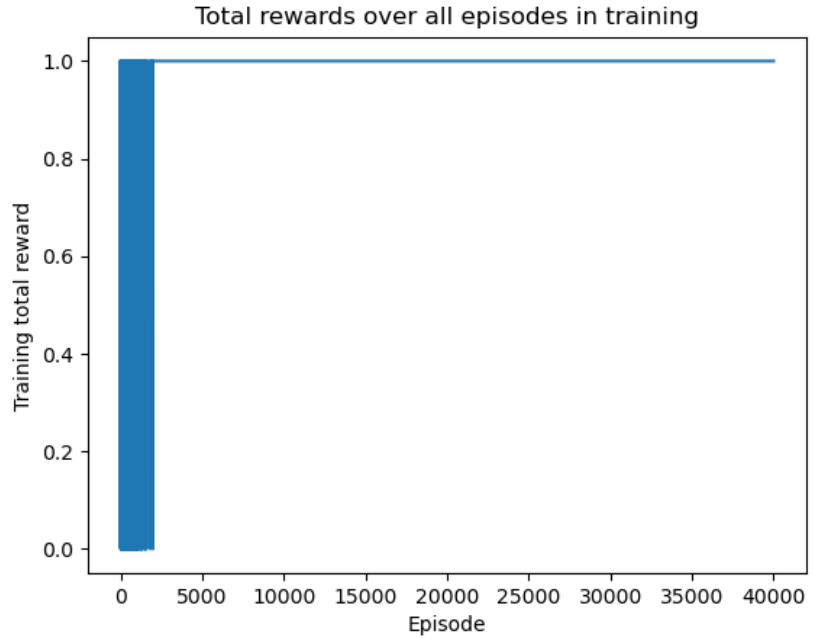
- You need to do self-study if you choose this technique to train your agent. Read this article for a practical guide to deep Q-networks.
  - With TensorFlow, build a model using Keras Dense layers.
  - In deep Q-learning, you need to make two models: main model and target model.
  - A replay memory (or buffer), storing “past experiences”, is needed and is usually implemented using a deque.
  - Take mini-batches from the replay memory as input-outputs to fit the main model; update the network weights using the Bellman equation. Copy weights to the target network at certain step intervals.

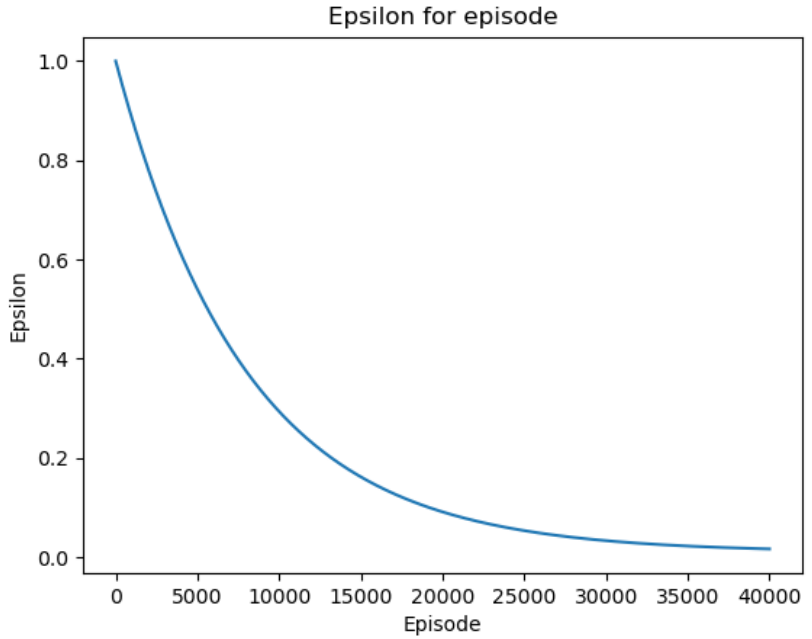
Deep Q-Learning

- Restrictions
  - Perhaps you may be aware of libraries such as TF-Agents, OpenAI Baselines and Stable Baselines. They contain some DQN implementations based on TensorFlow. For example, TF-Agents has tf_agents.agents.DqnAgent.
  - Using them can greatly simplify DQN programming. In this project, you are NOT allowed to use them, with an exception: if you have implemented Q-learning from scratch by yourself but you hope to explore more and compare DQN with Q-learning in this project, then you can use the TF's DqnAgent, say, to quickly make another agent, which is for comparison purpose. But this is beyond our demand, we will still mostly grade your Q-learning agent only. Perhaps doing so can benefit your report part (10%) since you can include more experimental results (more in-depth).
    - And note that in order to use TF-Agents, you need to convert your game's Gym environment into a PyEnvironment, typically using suite_gym.load(your Gym env). In this case, when you make an instance of your Gym env, you may need to avoid using wrappers such as TimeLimit, FlattenObservation, etc. which could cause troubles to the environment conversion.

Data Visualization

- Add some plotting code (using Matplotlib) to your project. For example, add it in your agent training script to plot the variations of some important parameters like reward or game score obtained per episode, the epsilon value used in your Epsilon-greedy algorithm, etc. You can plot them at the end of the training, or interactively, call `plt.ion()`, during the training process (with an option to disable it for faster training). Either way is acceptable.





Other Requirements

And reminders

Modularize Your Programs

- Suppose your game is called “Flappy Bird”.
- Separate your program functionality into different modules, e.g.:
  - `flappybird.py` – contains your PyGame Gym environment
  - `flappybird_play.py` – human play or random agent play
  - `flappybird_train.py` – train the AI agent
  - `flappybird_test.py` – test or use the AI agent
  - Other modules that contain classes of your game objects like enemies, power-ups, and other code like helper functions.

Command-Line Arguments

- Your programs should be written to accept command-line arguments for controlling parameters like:
  - Rendering mode (e.g., human, rgb_array)
  - Problem size (e.g., size of game board) if controllable
  - Seed for random number generator of your gym env. and action space (if any)
  - Number of episodes (for training, testing or playing)
  - Number of maximum steps per episode
  - Other possible parameters to tune: e.g., frame rate (fps), no. enemies to appear per time interval, which indirectly means the game's difficulty level
- Some references on argparse usage
  - A simple guide to command line arguments with argparse
  - Argparse documentation
  - Argparse tutorial

Packages Involved

- This project mainly involves the following packages:
  - pygame
  - gym
  - numpy
  - matplotlib
- Optionally, your project may use the following if you see fit:
  - tensorflow (v.2) [for DQN, don't use PyTorch this time]
  - pandas
  - Seaborn
  - Some other 3rd packages your game may need
- We assume you are using the latest or a recent version for each package.

Packages Involved

- Execute the following two commands in your virtual environment:

```
conda list -e > requirements_conda.txt
```

```
pip list --format=freeze > requirements_pip.txt
```

- Put these two output files in the root of your game folder.
- They are useful for us to reproduce the same runtime environment to grade your programs.

Beware of Copyright and Plagiarism

- Don't copy the source of another team.
- Don't copy the source of any existing game from the Web.
- You may enhance or revamp an existing game (found on GitHub, say) but in this case:
  - You must state the source (URL) clearly.
  - Your modification must be significant enough – at least 40% new source code.
  - If you would choose to publish your game to the public, beware that your act won't lead to any copyright infringement issues.

Important! Please read!

Acknowledgment of Resources Used

- Acknowledge in your source code the sources (including URLs) of all the images, sounds or video clips that you used in your game.
- Alternatively, you may list them all in your report or create a readme text file in the game folder.
- Note this game project is just for academic demonstration within the class only.
  - Generally, using copyrighted web resources in a class setting is considered "fair use" and acceptable.
- You cannot use your game for any commercial purpose if it has incorporated any licensed resources unless you have paid them.

A Sample Gym Environment

You can see a baseline of what we expect in this project

GridWorld Example

- The Gym documentation "Make your own custom environment" used the GridWorld example (GitHub link).
- We slightly modify this environment – change the state (or observation) from a dictionary to a numpy array of 4 integers.
  - Note: you could also use the FlattenObservation wrapper provided by Gym to do so in the client code.
- The modified gym environment is packaged as gym-boardgames.zip. Download this file from Blackboard and unzip. Run the following commands to install this package:

```
cd gym-boardgames
pip install -e .
```

  - Note: don't put this folder in a network drive; use local drive or else pip on Windows may encounter some weird problems in loading the package.

GridWorld Example

- Download gridworld-starter.zip from Blackboard and unzip it.
- Then you can run the programs in the folder:
  - `gridworld_play_v1.py` – create the Gym environment which can be played by human or by a random agent
  - `gridworld_play_v2.py` – using Gym's utility play script to make the environment playable by human.
  - `gridworld_qtest.py` – the code of the AI agent playing the game using the q-table model loaded from a text file using NumPy's loadtxt() function.

- The text files below store the training result (Q-table) for different board sizes:
  - `qtable_gridworld_5x5.txt`
  - `qtable_gridworld_10x10.txt`

GridWorld Example (Human Play)

- Random agent playing the game without GUI with default board size (5x5):

```
python gridworld_play_v1.py
```

- Random agent playing the game with GUI, board size of 10x10, at frame rate of 30 frames per second, for 10 episodes:

```
python gridworld_play_v1.py -m human_rand -n 10 -fps 30 -e 10
```

GridWorld Example (Random Agent Play)

- Random agent playing the game without GUI with default board size (5x5):

```
python gridworld_play_v1.py
```

- Random agent playing the game with GUI, board size of 10x10, at frame rate of 30 frames per second, for 10 episodes:

```
python gridworld_play_v1.py -m human_rand -n 10 -fps 30 -e 10
```

GridWorld Example (AI Agent Play)

- Q-learning agent playing the game without GUI with default board size (5x5):

```
python gridworld_qtest.py
```

- Q-learning agent playing the game with GUI with board size (10x10) at 30 frame/s for 10 episodes, with 500 max. steps per episode:

```
python gridworld_qtest.py -n 10 -m human -fps 30 -e 10 –ms 500
```

  - Without specifying the --file option, this script assumes the file name “qtable_gridworld_10x10.txt” for loading the q-table. You can load another file using the --file (or -f) option.

GridWorld Example (AI Agent Training)

- The training script `gridworld_qtrain.py` is not provided.
- But assume it is available, the following command is to train the Q-learning agent in a non-GUI mode (i.e., rgb_array or None) with a board size 10x10 for 40,000 episodes with 600 maximum steps allowed per episode, and the model will be saved as a text file tmp.txt in the current directory:

```
python gridworld_qtrain.py -f tmp.txt -n 10 -e 40000 -ms 600
```

- Non-GUI mode training is fastest. But the script also supports training with GUI mode to let users visualize the gameplay during training. One may observe an improving or smarter gameplay over time. Below command is an example. The training speed is surely much lower than non-GUI mode but you may use a higher fps to attain a better speed.

```
python gridworld_qtrain.py -m human -f tmp.txt -n 10 -fps 60
```

Grading

Grading Scheme (Tentative)

- Game code (60%)
  - GUI programming — 20%
    - Proper showing of game entities (players, enemies, score, buttons, …)
    - Proper use of multimedia (images, sounds, etc.)
    - Game over handling; game scene switching (if any)
  - Game interaction — 25%
    - Event handling, sprites, sprite groups, collision detection (if any)
  - Conformance to Gym environment — 10%
  - Command-line argument handling — 5%
- Agent code (30%)
  - Training the model — 15%
  - Using the model to drive gameplay — 5%
  - File I/O for saving and loading the model — 4%
  - Plotting curves of training results — 6%
- Report (10%)

Note: Without looking into your products and discussion with TA at that time, it is hard to finalize the grading scheme. So, we reserve the right to make minor adjustment to this tentative mark allocation after your submission.

Grading Scheme (Tentative)

- For each part of programming code, quality judgment or grading criteria include the following items (when applicable):
  - Completeness
  - Correctness
  - Code conciseness
  - Efficiency (particularly in agent training)
  - OO design (on the game code part only)
    - Proper use of abstraction, encapsulation, inheritance, polymorphism
  - Modularization
    - Proper decomposition of the whole into modules, classes, functions
  - Suitable documentation (docstrings, comments, or readme)

Some Hints

Multiple Game Scenes

- Only if you can manage to do so
- A game may consist of several scenes, for example,
  - A title scene
  - The main gameplay scene
  - A game-over scene
- Each has a different background and purpose.
- How to implement a game with multiple scenes?
- Check this out:
  - switch_scene.py in OtherSamples.zip

How to Show Text on the Screen?

- Check these out:
  - text_display.py in OtherSamples.zip
- Hint: to show a game score which gets updated from time to time, if you use a text surface to show it, every blit will produce a new text overlapping with old ones on the screen. You need to fill the background color or paint the background image again to hide those old text surfaces before the blit of the new ones. The sequence of such drawing actions matters.

How to Handle Mouse Events

- Check these out:
  - event_handler.py in OtherSamples.zip
  - image_click_test1/2/3.py in OtherSamples.zip

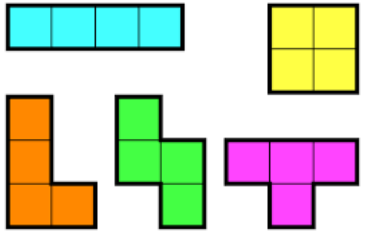
- Hint: Beware that for a surface, the method get_rect() returns a **new** rectangle covering the entire surface. This rectangle will always start at (0, 0) with a width and height the same size as the image.
  - Study image_click_test1.py, if you write the collision detection as "`if red_square.get_rect().collidepoint(x, y):`", it won't work as expected because the return rectangle always start at (0, 0).

Buttons

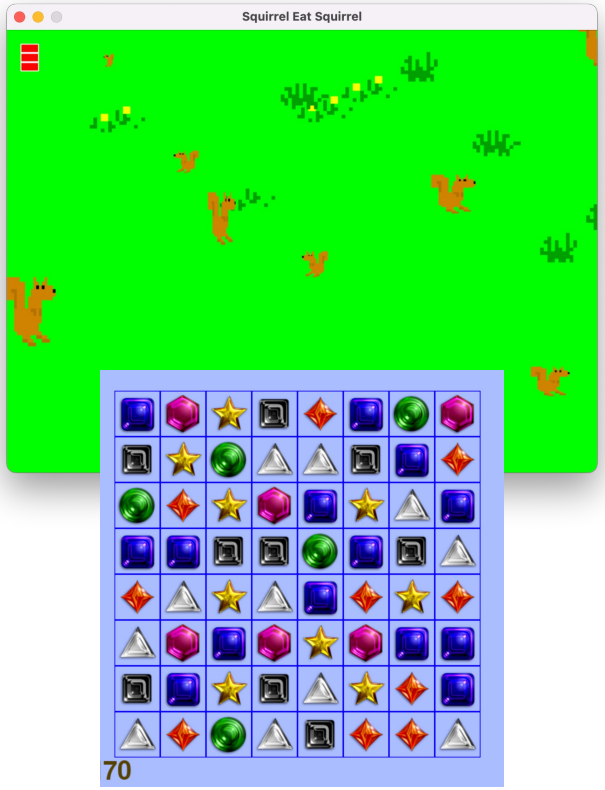
- There are usually some buttons in a game.
- When mouse-clicked (or selected by arrow keys and hit Enter via keyboard), a certain action is triggered, e.g., quite the game, or go to a certain game scene.
- Check these out:
  - image_click_test1/2/3.py in OtherSamples.zip
  - Other references: https://pythonprogramming.altervista.org/buttons-in-pygame/
    https://www.thepythoncode.com/article/make-a-button-using-pygame-in-python
  - A package of pygame buttons: https://pypi.org/project/pygame-button/

PyGame Samples and Some Game Ideas

PyGame Samples

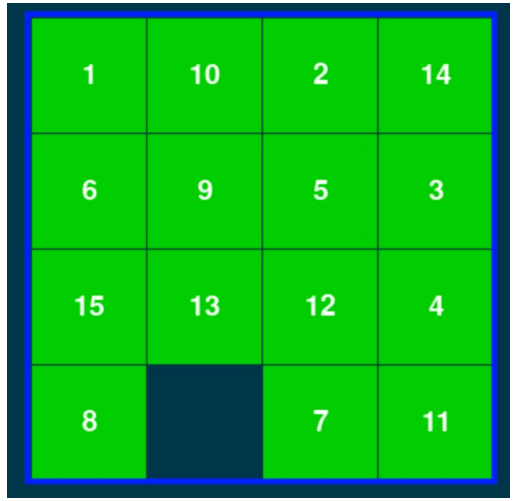


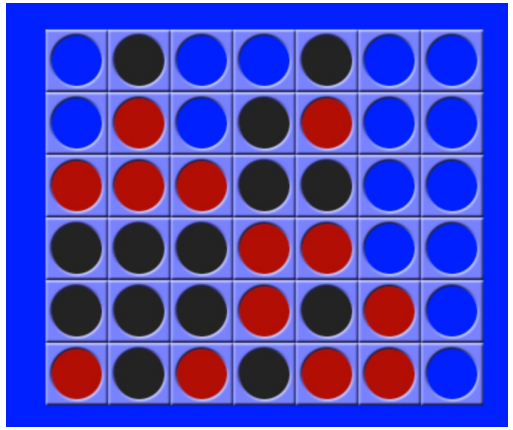
- Sample games from the reference book: https://inventwithpython.com/pygame/
- Downloadable as zip: https://inventwithpython.com/makinggames.zip
- Games:
  - Star Pusher: starpusher.py
  - Squirrel: squirrel.py
  - Tetromino: tetromino.py
  - Bejeweled: gemgem.py

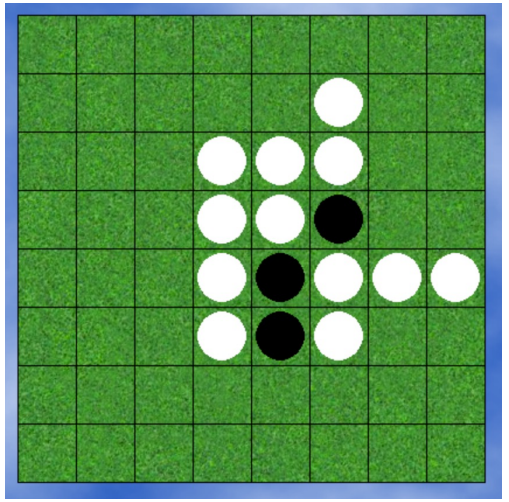


PyGame Samples

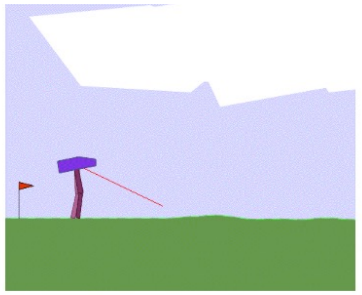
- Games (cont'd):
  - Slide Puzzle: slidepuzzle.py
  - Four in a Row: fourinarow.py
  - Reversi: flippy.py
  - Ink Spill: inkspill.py
  - Memory Puzzle: memorypuzzle.py



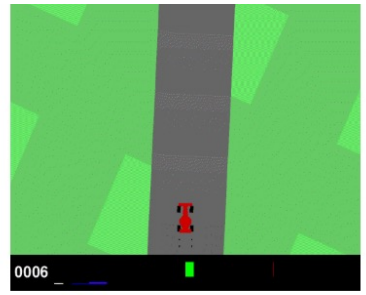




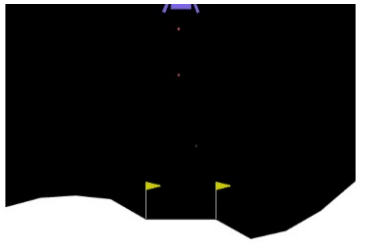
Gym Environments



Bipedal Walker

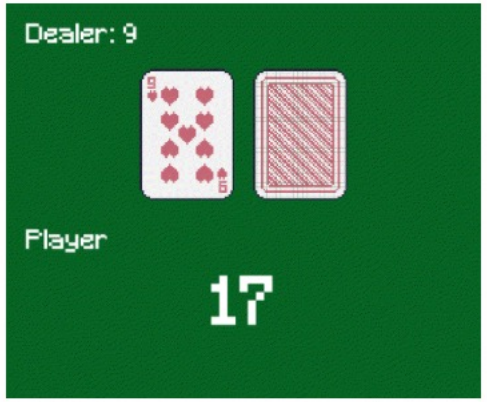


Car Racing



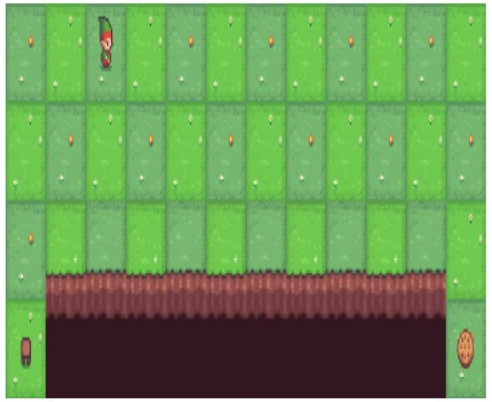
Lunar Lander

- https://www.gymlibrary.dev/environments/toy_text/
- https://www.gymlibrary.dev/environments/box2d/
- These environments also look like a game. You may consider using them as code starter and make an enhanced or revamped version.



Blackjack

Taxi

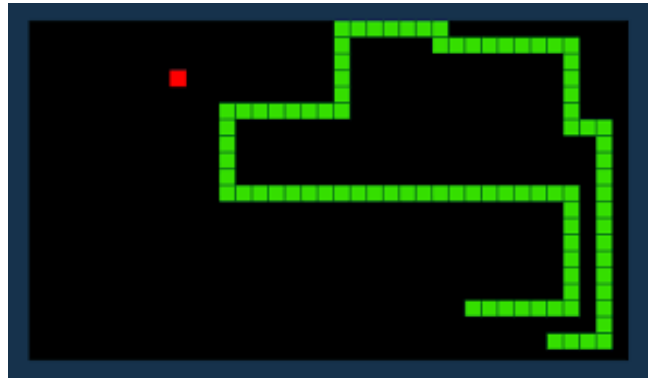


Cliff Walking

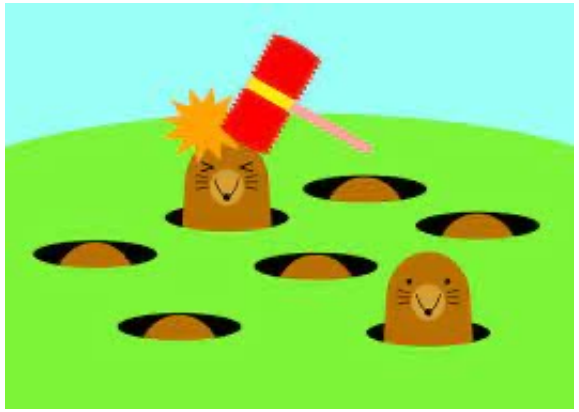
Frozen Lake

Game Ideas

The classical Snake game looks too much a cliché and its training can be found on the Web easily. Please avoid it (or upgrade the idea).



- Easier to implement:
  - Balloon pop or balloon shooter game
  - Whack-a-mole
  - Snake (or called wormy.py in the reference book)
  - Pong
  - Arkanoid
  - Tic tac toe
  - Sudoku
- Can you enhance them?

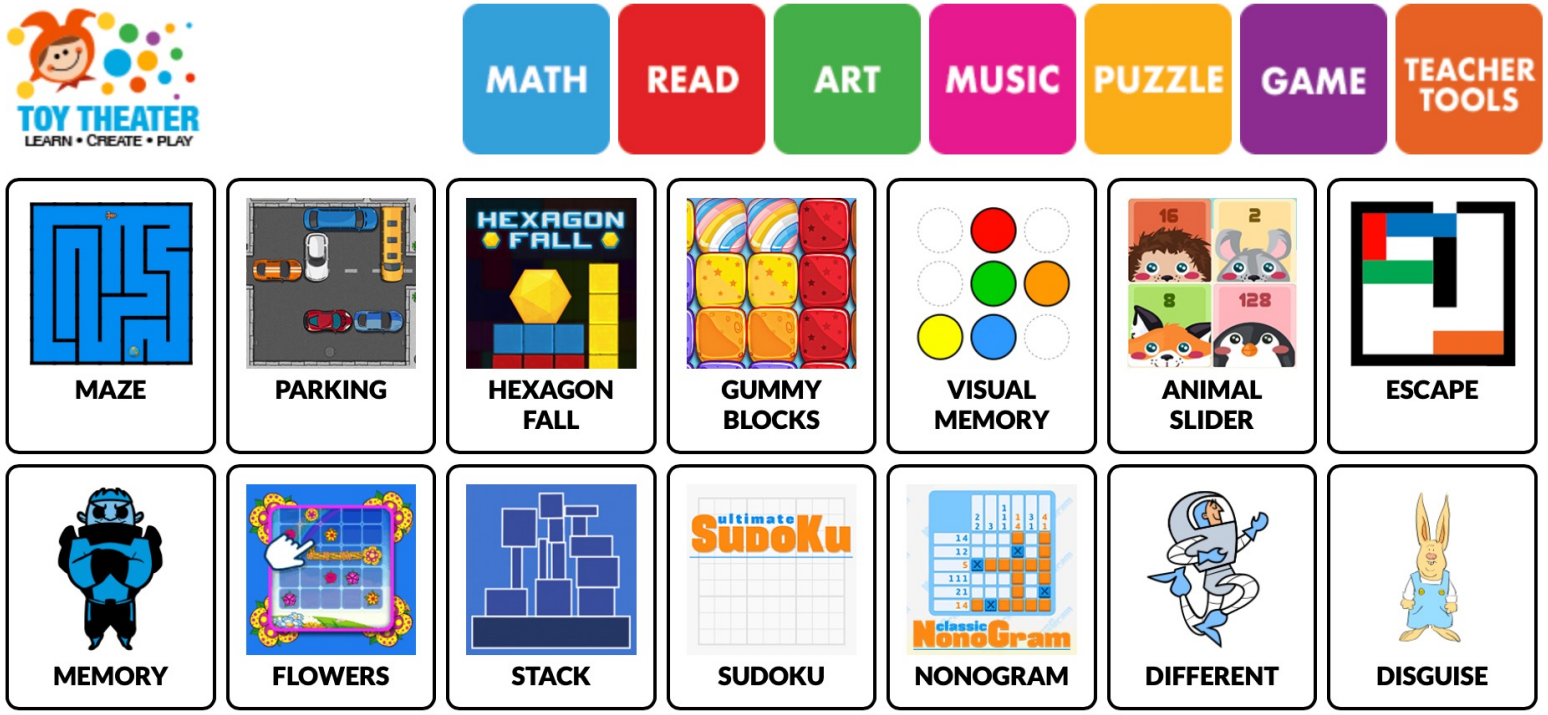


Game Ideas

Websites like these have many games that are relatively easy to implement. Browse for more ideas.

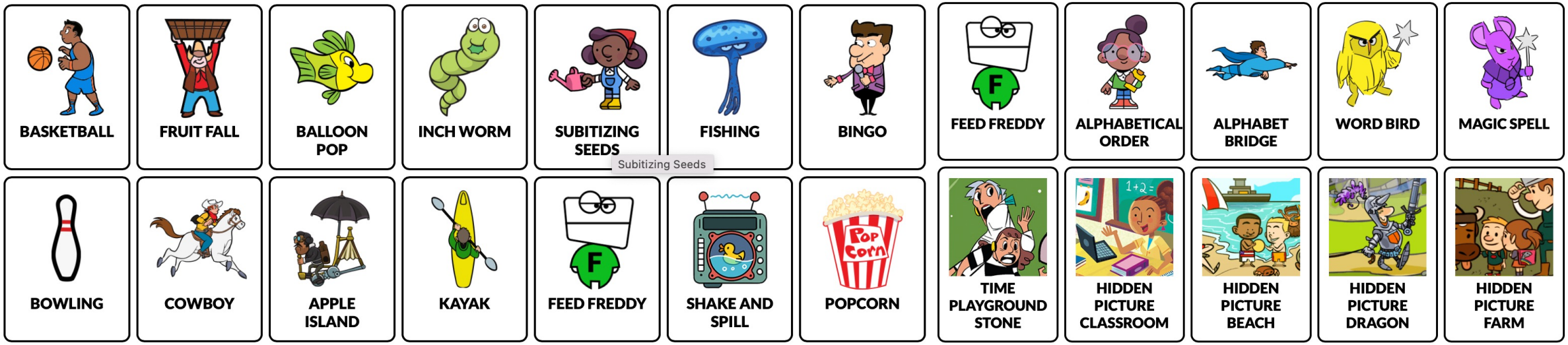
- Educational games for kids
  - https://toytheater.com/
  - https://wordwall.net/en-us/community/games

https://www.puzzleplayground.com
https://www.mathplayground.com



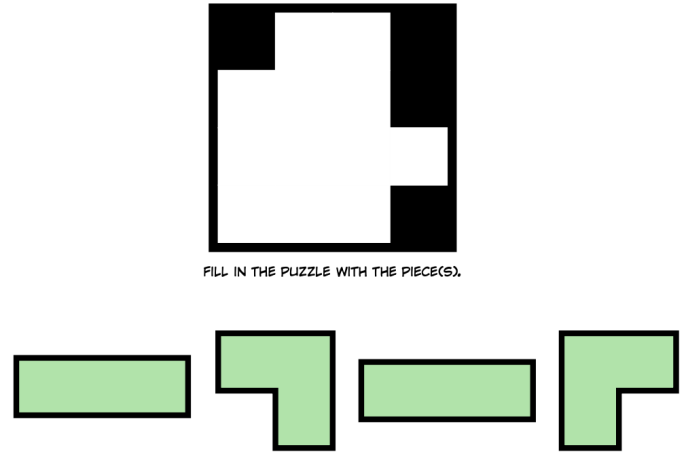
Game Ideas

- https://toytheater.com/



Game Ideas

- These games look good.

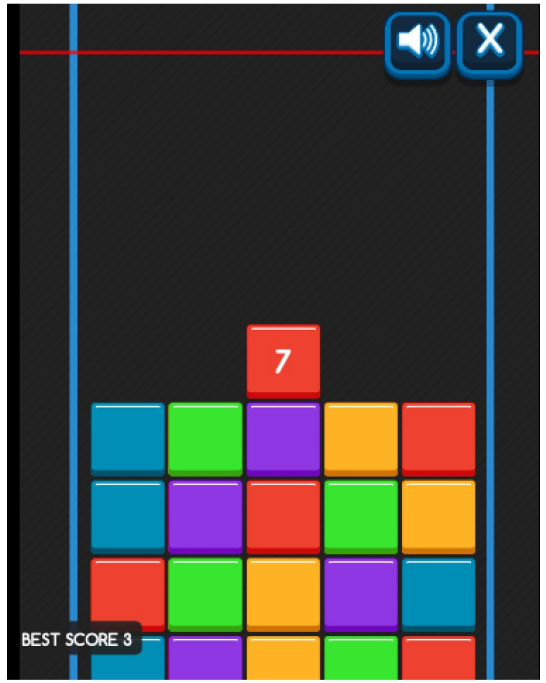


https://toytheater.com/pieces/

https://toytheater.com/parking/

https://toytheater.com/flowers/

https://toytheater.com/sliding-squares/



Game Ideas

- Convert some board games or card games to computer games, e.g.,
  - MasterMind (online version)
  - Scrabble
  - Gekitai (online version)
  - Uno (online version)
  - …

- (But don’t copy source from any GitHub.)
- You may skip the AI part first and make it a 2- or n-player game.
- Then build the AI agent and make it as a computer player.

Game Ideas

"Cut the rope" involves *physics* (*gravity*). There is a simple Python package to ease the programming. Check these out: Repl.it and pypi.org.

- Harder: can you make a game like these by PyGame?
  - Royal match
  - Cut the rope
  - Blockudoku
- Check out some online game websites for more ideas?

  https://www.crazygames.com
- Add your imagination!

There is a GitHub repo implementing Blockudoku using PyGame. But it looks not good enough. See if you write a better one

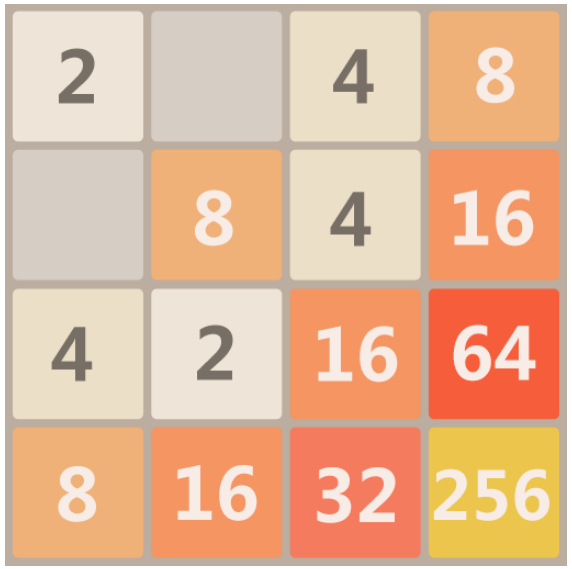
If you look for a jewel matching game built on PyGame, gemgem.py in the reference book is a good example.

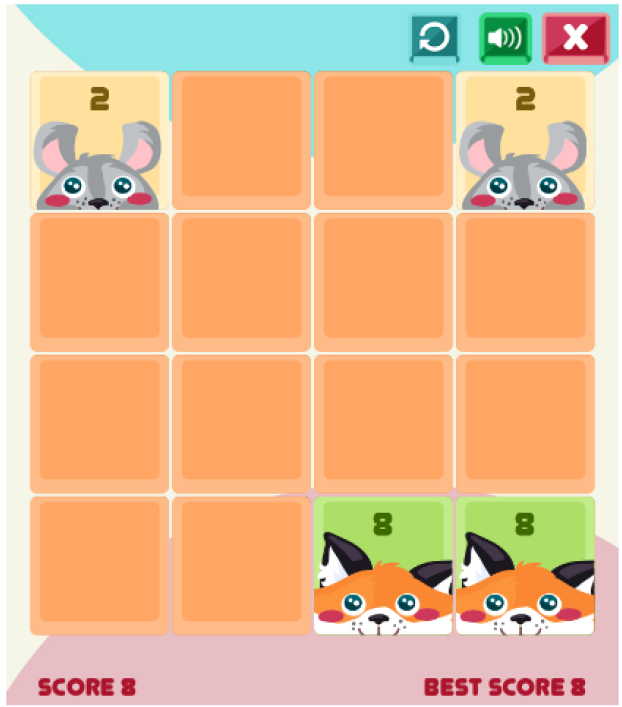
Game Ideas

- Variants of the 2048 game:
  - Original version – source code available
  - Drop the Number – you can implement this.
- You may enhance the game in graphical design and add new control (mouse).

PyGame source code:
https://github.com/rajitbanerjee/2048-pygame

https://toytheater.com/animal-slider/





Full List of (Game) Projects on pygame.org

- Check this out: https://www.pygame.org/tags/all
- Click the different tags like arcade, puzzle, shooter, platformer, rpg, action, adventure, retro to find game projects of the desired genre.
- They usually provide source code and resource files.

Useful Resources

Free Game Resources

- Here are some sources for music, sound, and art that you can search for useful content:
- **OpenGameArt.org:** sounds, sound effects, sprites, and other artworks
- **Kenney.nl:** sounds, sound effects, sprites, and other artworks
- **Gamer Art 2D:** sprites and other artworks
- **CC Mixter:** sounds and sound effects
- **Freesound:** sounds and sound effects

More info:
https://inventwithpython.com/blog/2011/04/30/free-music-sound-effects-tiles-and-2d-art-to-use-in-your-games/

Free Game Resources

If you want to make a car racing game, this game kit can save some effort.

- FreeSFX: sound effects
- SoundBible: sound effects
- Craftpix.net (Freebies): sprites, characters, backgrounds, tilesets
- itch.io (free): sprites, characters, backgrounds, tilesets
- Reddit /r/GameAssets: free game assets shared by the community

More info:
https://blog.felgo.com/game-resources/16-sites-featuring-free-game-graphics

More Tutorials on PyGame

- https://www.pygame.org/wiki/tutorials
  - The tutorial on specific topics are quite useful.
- https://www.geeksforgeeks.org/pygame-tutorial/

- Top 5 Pygame Open Source Projects in 2022 [For Beginners & Experienced]
  - https://www.upgrad.com/blog/pygame-open-source-projects/