Project Documentation of

Tetris Game

Ken Shanyi ZHANG (sz1851@nyu.edu)

Qingyang LI (ql2048@nyu.edu)

CSCI-GA-2440 Software Engineering (Fall 2021)

Prof. Lihua Xu

Courant Institute of Mathematical Sciences

New York University

December, 2021

Table of Contents

Introduction

This article is the documentation of our final project in the Software Engineering course. The project is the Python implementation of Tetris Game.

1) **Tetris Game**
   Tetris Game is a puzzle video game created in the 1980s. It is based on simple rules yet requires skills and intelligence, which makes it one of the most popular early video games in the world.

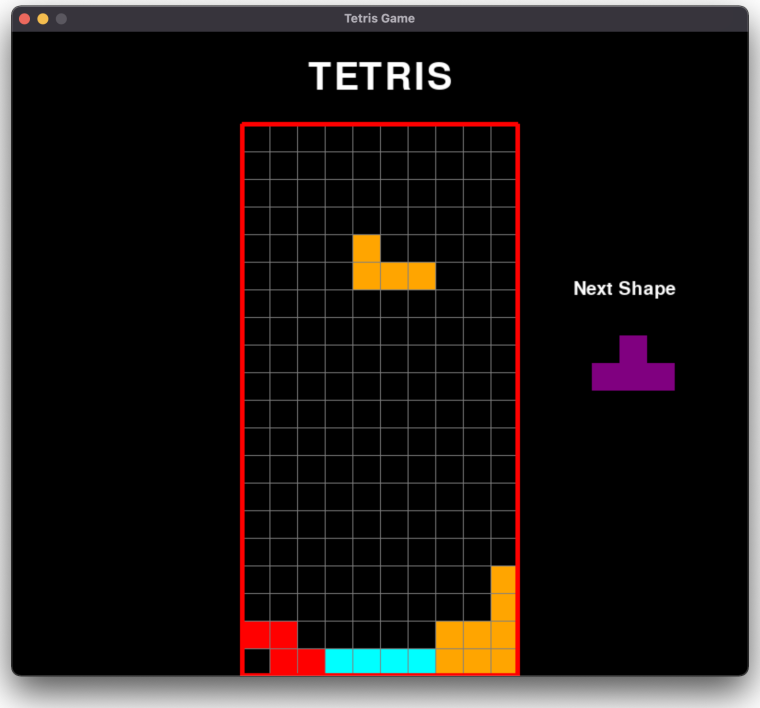

Figure 1 User Interface of Our Tetris Game

2) **Play Rules**
   Basically, players need to complete rows by moving different Tetrominoes. Score that a player can get is proportional to the number of completed lines. At the beginning of the game, you can enter your username and set up the level of difficulty. You use *up* key to rotate a Tetromino, *left* key to move left and *right* key to move right. *Down* key is used to accelerate the Tetromino descent. The right panel shows the Tetromino that is about to appear next. You lose if any piece gets stacked up and reaches the top of the grid. The longer the player can delay this outcome, higher score is expected.

3) **Key Features**
   Our program enjoys the following features: 1). Inspired by Generative Adversarial Network (GAN) [1], there will be a model (either a multi-layer perceptron or a deterministic predictor) that acts as an opponent obstructing the player to win. In detail, the upcoming Tetromino is determined by this model based on the current state of the grid. The behaviors of these two model candidates are quite different. Perception will get penalized if the player successfully eliminates rows using the Tetromino it generates. Thus, it will evolve and eventually generate Tetrominos with non-uniform contours (typically T-shape and Z-shape ones) repeatedly to cause the player headache. In contrast, the deterministic predictor performs constantly,

which will ban a certain Tetromino if the corresponding grid pattern occurs (e.g. T-shape Tetromino will be banned if there is a single slot to complete a row). 2). The program enables customization of level of difficulty and username. At the very beginning of the game, the player is allowed to select the level of difficulty and type in the username. The level of difficulty will determine the falling speed of Tetromino and exploration (generate the next Tetromino at random) / exploitation (generate the next Tetromino using the model) ratio. At the end of a game, the username-score pair will be stored and displayed in the leaderboard. 3). Thanks to the PyGame framework and Python Virtual Machine (PVM), this program is cross-platform and device-agnostic. 4). The general behavior of this program is governed by several global parameters (window size, granularity of grid, etc.). We tried our best avoiding hard-coded configuration.

Framework

The frameworks that we employ for developing this game are: SQLite[2], PyGame[3] and PyTorch[4]. SQLite is a widely-used relational database management system embedded into end programs. We apply SQLite as it generally follows the protocol of PostgreSQL and provides bindings to Python. PyGame is a Python module designed for developing video games. We use PyGame because: 1). It is a cross platform software. 2). It provides nice toolkits (graphical user interface, user event handle etc.). 3). It's compatible with the complexity of our application. PyTorch is one of the most popular machine learning frameworks developed by Facebook (now Meta Platform) AI. Two key characteristics of Pytorch are 1). It enables GPU acceleration. 2). Back-propagation is computed automatically as you build the forward pass. In this work, we use PyTorch to construct a multi-layer perceptron to generate the upcoming Tetromino that is empirically least likely to let the player gain scores.

Development Process

Week 1-5: Apply Software Engineering Basics into Project

1. Class/Sequence Diagram
2. Requirement Specification
3. Use Case Diagram

Key Point: Brainstorm a reasonable topic

Week 6: **Agile Planning**

1. Formulate Progress Report for each iteration
2. Framework selection and setup (PyGame, Pytorch, etc.)
3. Design tentative class API interfaces

Key Point: Come up with a practical roadmap to achieve our final goal

Week 7: **Tetromino**

1. Implementation of `Tetromino`
2. Write Unit Tests for `Tetromino`

Key Point: Design a legitimate representation of `Tetromino` shapes

Week 8: **Tetromino Proxy**

1. Implementation of `TetrominoProxy`
2. Write Unit Tests for `TetrominoProxy`

Key Point: Wrapper class to address commands from other modules

Week 9: **Main Function, User Interface & Game Board**

1. Design the `main()` function and `Gameboard`
2. Implement `UserInterface` backboned by `Gameboard`

Key Point: Finish the game workflow w/o actions from player

Week 10: **Main Function, User Interface & Game Board (cont.)**

1. Refactor `GameBoard` and `UserInterface` to handle user actions
2. Write Unit Tests for `GameBoard` and `UserInterface`

Key Point: Rewrite code in object-oriented fashion and rectify class methods

Week 12: **Leaderboard**

1. Implementation of `Leaderboard`
2. Write Unit Tests for `Leaderboard`

Key Point: Employ light-weighted SQL database to store player-score pairs

Week 13: **Enhancement of User Interface**

1. Add `InputBox` and `CheckBox` called by `UserInterface`
2. Write Unit Tests for `InputBox` and `CheckBox`

Key Point: Add interface for player to customize the game settings

Week 14: **Adversarial Learning for Tetromino Generator**

1. Add class `CNNModel` and `DeterministicModel` to achieve adversarial learning against the player
2. Write Unit Tests for `CNNModel` and `DeterministicModel`

Key Point: Generate upcoming `Tetromino` based on current state of `Gameboard`

Week 15: **Integration Test & Miscellaneous Items**

1. Integration testing
2. Documentation & Presentation

Key Point: Write well-designed documentation according to course requirements

Requirements & Specifications

The goal of this project is to implement a Tetris game using Python. Players may interact with the system through a Graphical User Interface (GUI), operate the Tetrominos through keyboard, and gain points based on their performance. The system operates the game following widely-accepted rules of Tetris mentioned above.

1) Use Cases

This system of Tetris game consists of the following use cases:

- UC1: Set Up Game
- UC2: View Leaderboard
- UC3: Move Downward
- UC4: Shift Horizontally
- UC5: Rotate
- UC6: View Next Tetromino

2) Detailed Description

UC1: Set Up Game
Right after the game starts, the Game Setup dialog will prompt the player to select the level of difficulty (that controls falling speed, Tetromino generation setup, etc.) for this game through a CheckBox and the name of the player through an InputBox. After that, the player may press Enter to start the game.
UC2: View Leaderboard
Right after the player finishes the game, the View Leaderboard dialog will prompt the player to view his/her score and the historical scores for this game. The player may press any key to close the Leaderboard window and get back to the main menu.
UC3: Move Downward
The game space is basically a rectangle with grids. There is always a Tetromino on the fly. The game is clock-based. Every time the clock ticks, the Tetromino will move one cell downward. The player can perform valid operations (see UC4 & UC5) between two consecutive clock ticks. Alternatively, the player can also press the *down* key to speed up the downward movement. This action will keep going on unless the current Tetromino hits the ground.
UC4: Shift Horizontally
The player may move the current Tetromino left/right by one grid by pressing the *left/right* key. After that, the horizontal location of the current Tetromino is altered, and the Tetromino continues to move downward.
UC5: Rotate
The player may rotate the current Tetromino by 90 degrees clockwise by pressing the *up* key. After that, the spatial position of the current Tetromino is altered, and the Tetromino continues to move downward.
UC6: View Next Tetromino
While the current Tetromino is moving downward, the player may check the current state of the game (e.g. location of the current Tetromino, shape of next Tetromino) and decide if further actions need to be taken.

3) Non-functional Requirements
NR1. Performance
The program should keep accepting the player's inputs during the game, but execute only the valid operations given a player's input to the system. All operations shall be completed quickly.
NR1.1 User Response
The program shall respond to any user input within 0.01 seconds.
NR2. Usability
Based on the GUI designed, a player should be able to determine quickly what type of operations they can perform.
NR2.1 Player options
A player shall only have access to functionality that is allowed at a given time.
NR2.2 User Interface
The program shall allow a player to interact with it through keyboard and mouse. During the game, all operations can be achieved using 4 Arrow keys of the keyboard. The amount of mouse and keyboard input shall be minimized by the system to include only selecting level of difficulty, entering username and closing windows.
NR2.3. User Errors
The program shall catch invalid input from all fields in the system.

4) Constraints
All code development shall be done with the Python3 programming language.
All testing shall be done using PyUnit.

5) Development and Target Platforms
Operating Systems: Windows 10, MacOS Monterey

Hardware Architecture: Intel Silicon x86

Architecture & Design

1) Class Diagram

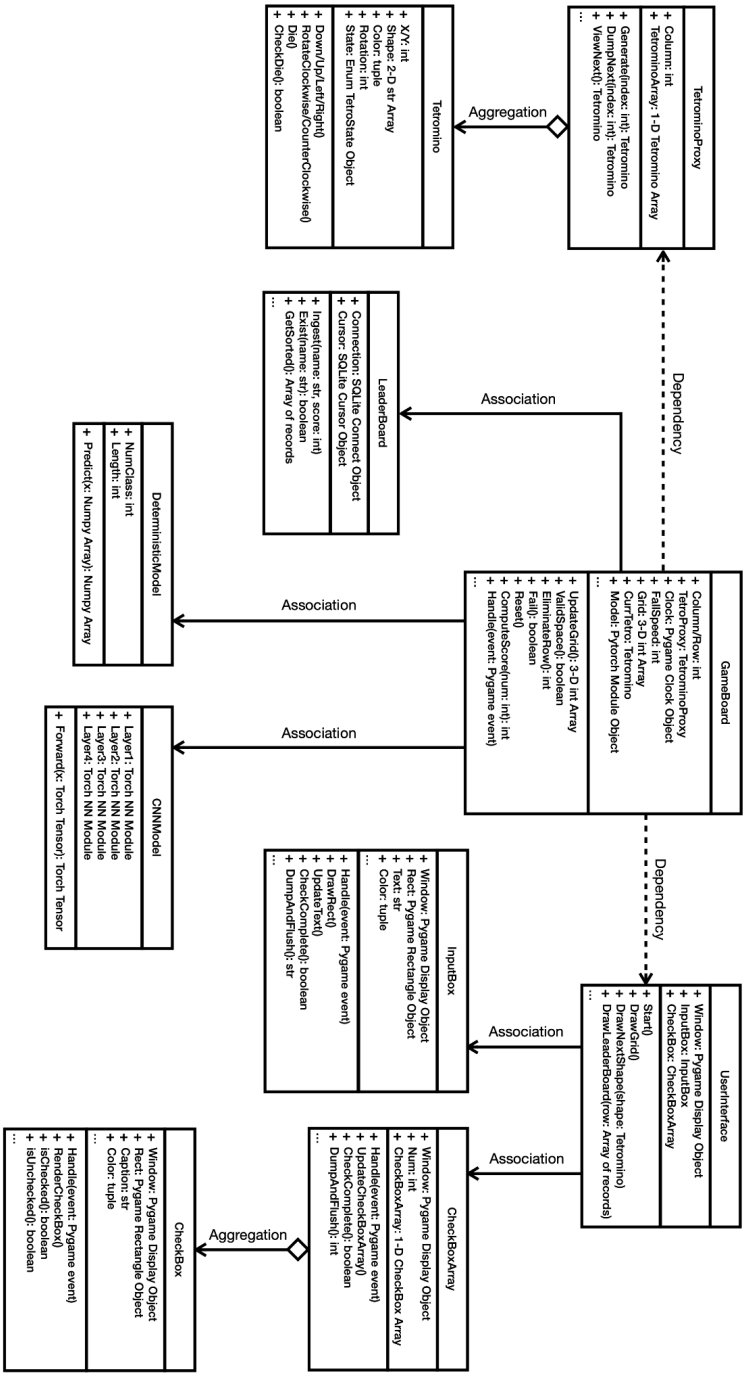

Figure 2 Class Diagram of Our Program

Observation Our program is designed according to the following philosophy: 1). `Gameboard` is designed as the controller to execute the whole workflow and record state transitions. It is responsible for triggering all other modules when needed. 2). The internal mechanism is detached from the user interface. All functionalities regarding display are implemented in `UserInterface`. 3). `TetrominoProxy` is mainly a `Tetromino` buffer handling request from `Gameboard` and `UserInterface`. 4). `InputBox` and `CheckBoxArray` (aggregation of `CheckBox`) are two components used by `UserInterface`. 5). `LeaderBoard` is backboned by SQLite to store user-score pairs. 6). `CNNModel` is simply the implementation of the multi-layer perceptron, while `DeterministicModel` is implementation of the predictor based on grid pattern detection.

2) Sequence Diagram

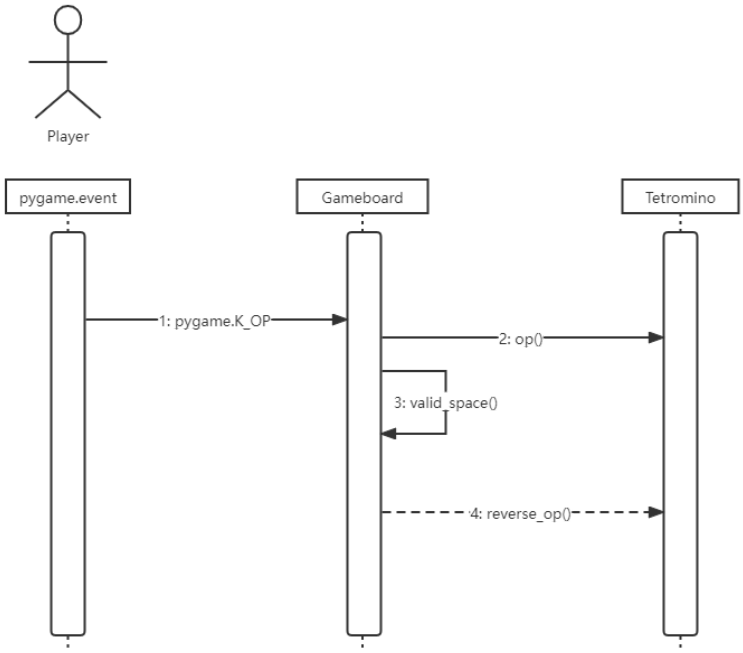

Figure 3 Sequence Diagram for Operations

Observation In PyGame, all user actions are captured by `pygame.event()`. In our design, they are passed to the event handle of `Gameboard`. `Gameboard` will first perform the corresponding action on the current `Tetromino`. It will then check the validity of this action by calling `valid_space()`. If this is NOT a valid action, `Gameboard` will perform the reverse action to cancel it out.
The potential `op()` and their `reverse_op()` can be:

Table 1 Valid Operations for Tetromino

`op()`	`reverse_op()`
left	right

right	left
clockwise	counter-clockwise
counter-clockwise	clockwise

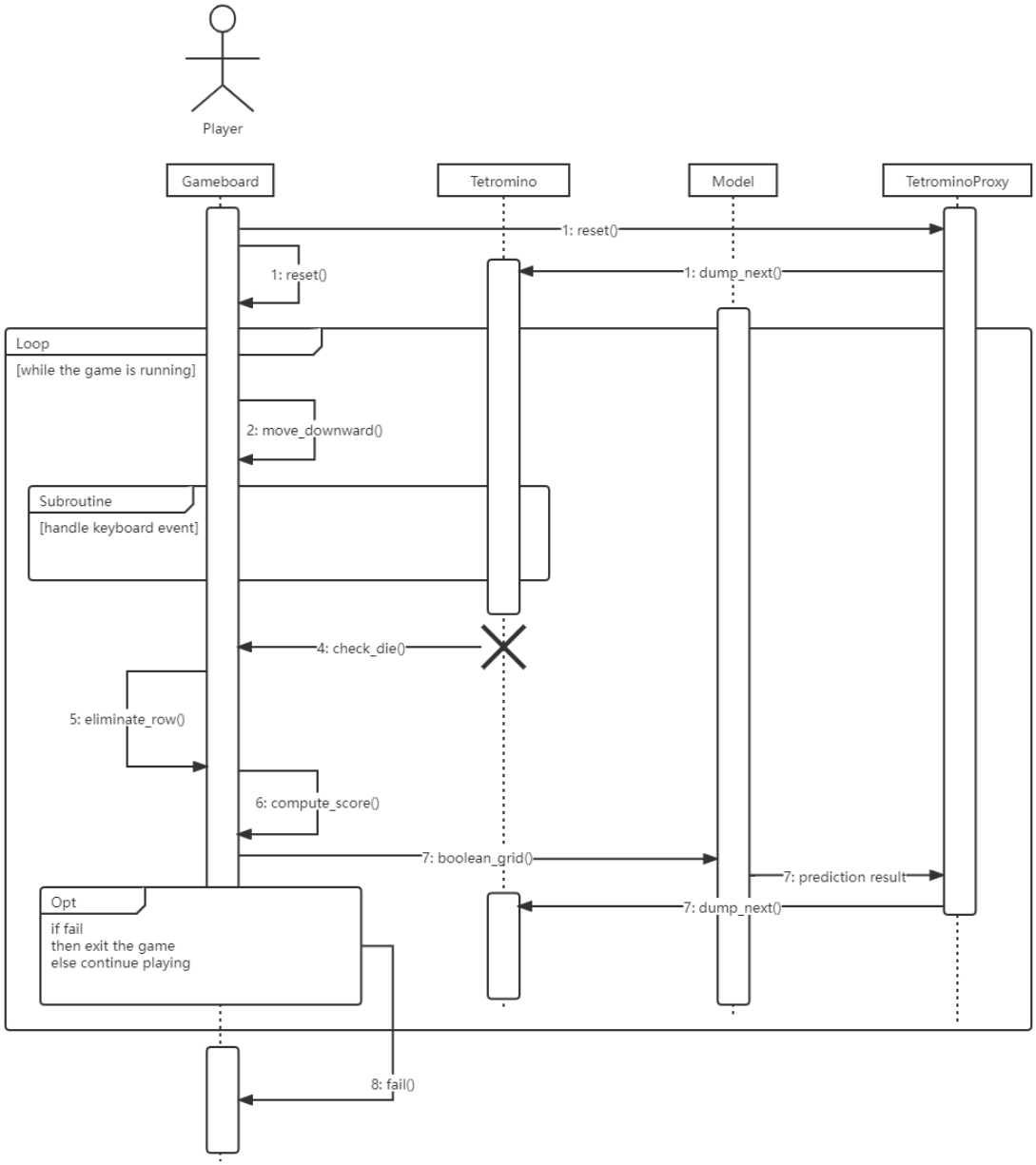

Figure 4 Sequence Diagram for Playing the Game

Observation To start the game, `Gameboard` will reset its state and the `TetrominoProxy`, and an initial `Tetromino` will be dumped from the `TetrominoProxy`. The game process can be regarded as a loop. `Gameboard` moves the current `Tetromino` one cell downward after each clock tick. The player may perform “shift left”, “shift right”, “rotate clockwise” and “rotate counter-clockwise” to change the location and/or orientation of the `Tetromino`. If the current `Tetromino` was detected “dead” (i.e., hit the

ground), `Gameboard` would try to eliminate rows filled with non-empty cells, and compute the score based on the number of rows eliminated. Meanwhile, the next `Tetromino` will be dumped from the `TetrominoProxy` following the request from `Model`. `Gameboard` finally checks if the condition of failure is fulfilled, and exit the game if it does.

3) Test Case Design

We use PyUnit, the default Python Unit Testing Framework, to conduct software testing. We group our test cases by 4 classes:

- `TestDisplay`: Test cases related to GUI display function powered by PyGame. The correctness of related modules (`InputBox, CheckBox, LeaderBoard`) is also verified.
- `TestModel`: Test cases related to the models used for adversarial Tetromino generation. Notice that this is an experimental feature, so we only test its correctness.
- `TestTetromino`: test cases for `TetrominoProxy`, with a focus on the movement (down, up, left, right, rotate) of a `Tetromino`.
- `TestGameBoard`: test cases for `GameBoard`. It includes unit test cases for all the important methods of class `GameBoard`, and an integration test case `test_play_noop()`.

Reflections & Lesson Learned

We can tell we learn a lot from this course project. We are thankful to the instructor and teaching assistant for their helpful advice.

1) Bug Resolving

Input box blinking is a nasty bug that we managed to conquer. In detail, the "symptom" is that the input box and the text will blink at an unreasonable refresh rate no matter what you type in. We did the following to resolve this issue: 1). Initially, `InputBox` is an independent module to capture input text, which causes the refresh rate inconsistency between itself and other components in the `UserInterface`. Therefore, we modify it as an attribute of `UserInterface` and share the same display update command with others. 2). The event array captured by the PyGame framework is too fine-grained for the input box. In this regard, we manually lower the frequency of for-loop by calling sleep function.

2) Code Refactoring

In the early version of our program, all modules related to display are functions that share a common parameter `pygame.window`. Moreover, they are entangled with `GameBoard`, which makes it impossible to write unit tests for `GameBoard` and interface modules individually. Therefore, we construct a class encapsulating all functionalities related to display and make `pygame.window` an attribute at initialization. This class also provides unified handles for `GameBoard` to address GUI issues. In this way, we draw a clear line between internal mechanism and front-end interface.
Another code refactoring that we implemented is the distribution of attributes among `GameBoard` and `Tetromino`. Finally, we adopt the suggestion from our teaching assistant that all attributes related to operations (e.g. x-y coordinate, rotation, …) are stored in `Tetromino` and those related to grid (e.g. occupied cells, clock, etc.) belong to `GameBoard`.

3) Limitations of Framework

At the beginning, we planned to use Tkinter as the GUI of our LeaderBorad, since it is natively installed with Python, and provides useful features for displaying tables. However, we found later that the newest version of MacOS doesn't support Tkinter, and the LeaderBoard window didn't pop up. Although there are ways to resolve this problem by reinstalling Python, we chose to switch back to the display feature of PyGame to improve the consistency of user experience and avoid the limitation of Tkinter.

Miscellaneous

In this section, we articulate several concerns that are required to be documented.

1) Code Comments

The source code of the program is heavily annotated. The command to generate the code documentation: $ python3 -m pydoc <file_name>

2) Installation & Distribution

In order to run our program smoothly, the following software packages should be installed: Python 3.6, Pytorch 1.8.1, Numpy 1.20.1, Pygame 2.0.2. Note that all other packages (e.g. Unittest, SQLite, etc.) are internally supported by Python.

3) Resources

Our public Github repository lives in: https://github.com/shanyizhang/SE-Proj

Reference

[1] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. Generative adversarial nets. In Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2 (NIPS'14). MIT Press, Cambridge, MA, USA, 2672–2680.

[2] SQLite Home Page, Access: https://www.sqlite.org/index.html

[3] Pygame, Access: https://www.pygame.org/news

[4] Pytorch, Access: https://pytorch.org/