Mario Bomber

CS 110 - Python Final Project
Pygame MVC Arcade
Alex Eskenazi and Vishil Patel

Team Members

- Alex Eskenazi - Project Lead and Developer
- Vishil Patel - Developer and Editor

Project

- Our project is a Python arcade game using Pygame with a MVC (Model/View/Controller) architecture

Libraries and tools

- Pygame - graphing event framework.
- Shelve - file data storage for data permanence.
- Os - operating system helper functions.
- Random - pseudo-random generator
- Pyreverse - analyzes your source code and generates package and class diagrams.
- Tabletomarkdown - converts spreadsheets into markdown for the readme.md file

Project - Mario Bomber

- Mario bomber is a game where bombs and other items rain down from the sky
- Mario, the character, has to avoid the harmful bombs raining from the sky.
- Raining mushrooms give Mario an extra life, but make the game go faster.
- The longer the character stays alive the more points it gets.

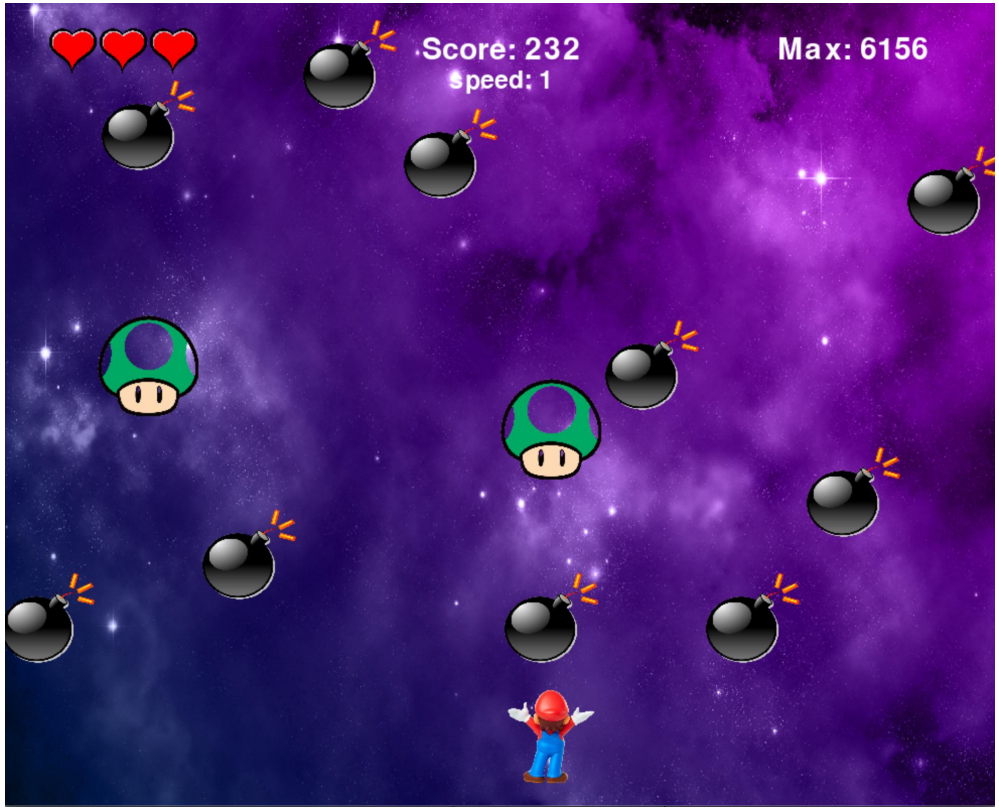

3000 pts
Front End Design -
Original GUI Concept

Front End Design - Final GUI Result

- Images:
  - Images came from across the internet and were cleaned using https://www.remove.bg/upload.
  - We obtained most images while brainstorming about what game to build
- Sounds:
  - Action sounds came from https://www.soundjay.com/misc-sounds.html
  - Background Soundtrack from https://www.soundjay.com/free-music-7.html
- Changes:
  - We stayed faithful to the original design for the most part, but did some modifications based on new ideas as we worked on the project.
  - We also had to do some minor changes to make the game more usable like placing the game objects on a grid so it would be a bit more predictable to the user.

Front End Design - Final GUI - Welcome Screen

Front End Design - Final GUI - Game Screen

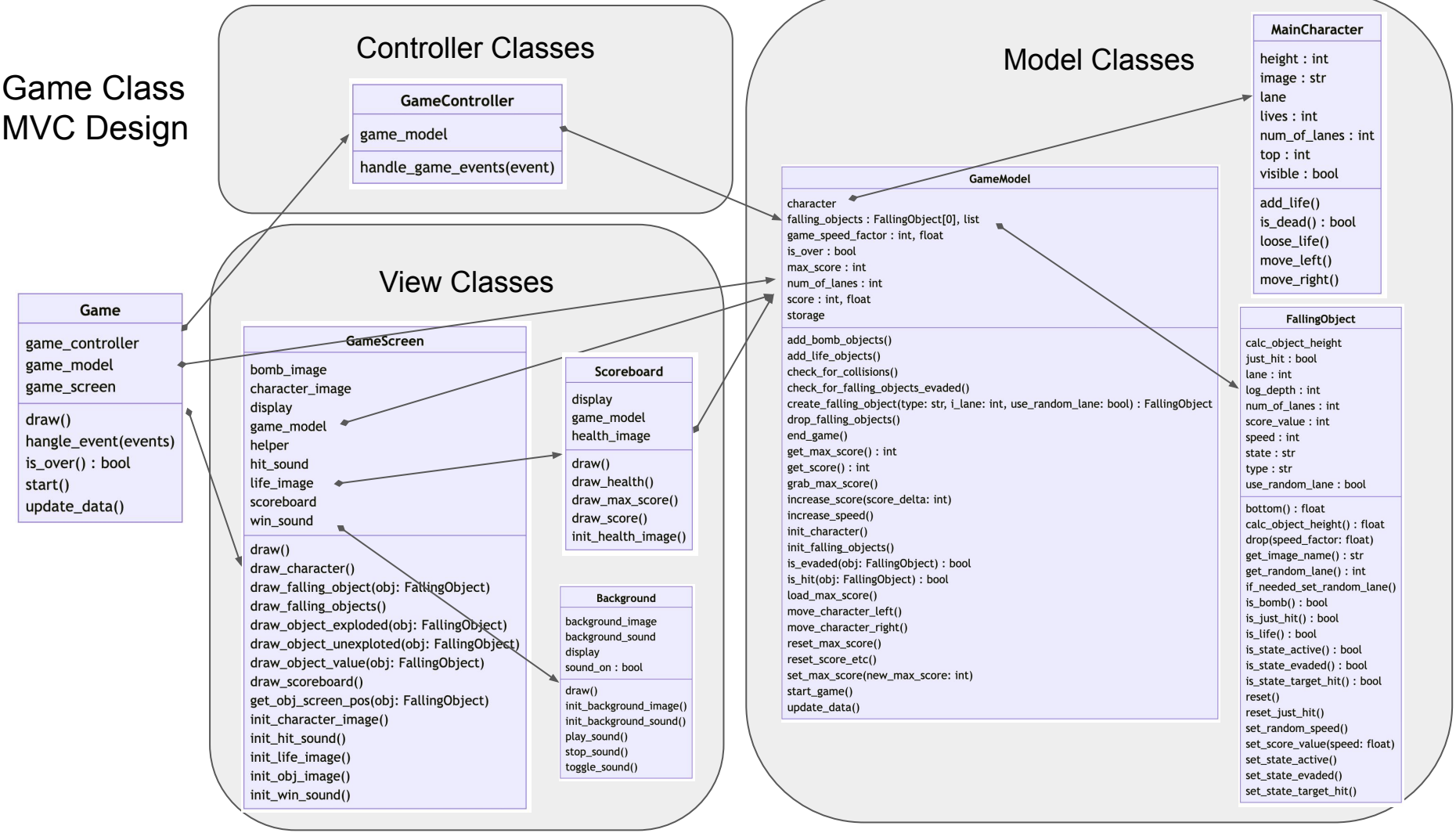


Front End Design - Final GUI - Game Over Screen

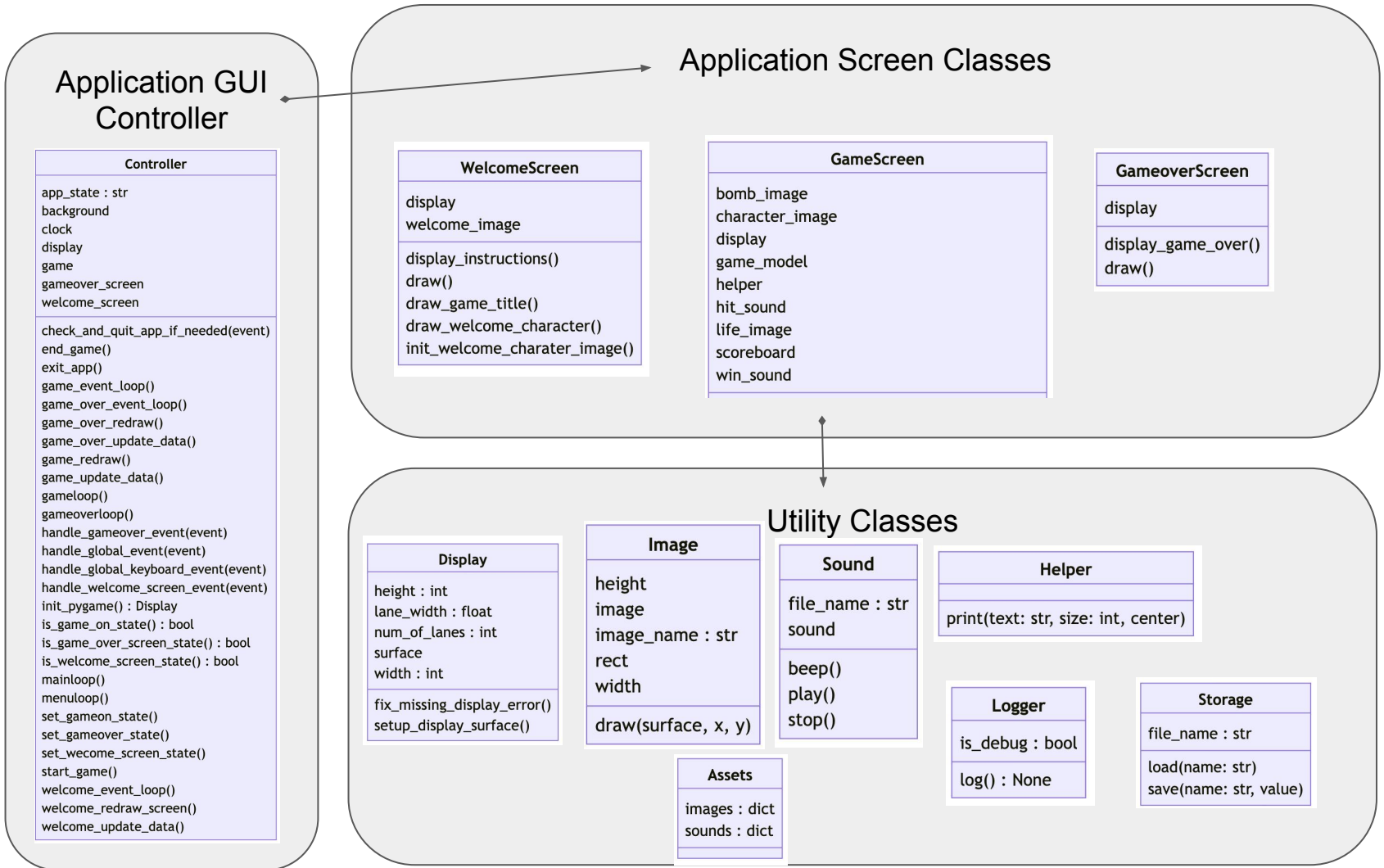
Front End Design - Biggest Front End Challenge

- Coming up with a logical screen model that was solved with logical screen units and lanes:
  - Lanes - We ended up using a grid which comprised of vertical lanes where the object reside and this helped avoid overlapping objects and helped with the debugging. It also gives a retro feel to the game.
  - Logical Screen Units - We created the model based on logical units that total 100 hundred units vertically and horizontally and then convert to screen coordinates in the View. This helped debug and make the game work on different screen sizes.
- Speed - tuning the speed of the game was challenged because it runs a different speeds on local laptops as compared to replit.
- Encapsulating pygame functionality for easy reuse (DRY) - solved by creating wrapper classes for text, images, and sound output.

Game Class MVC Design

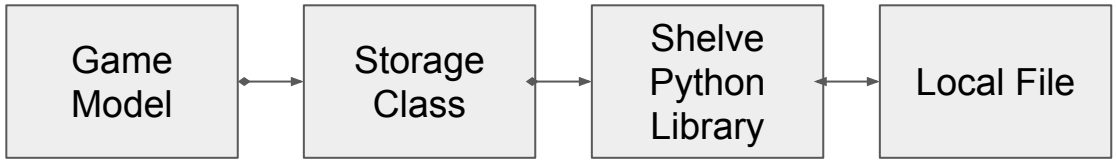


Application Class Diagram



Back End Design - Data Permanence

- Highest Score - The game stores and retrieves the highest score.
- Storage - the data is stored as key/value pair (dictionary) on a local file.
- Shelve - shelve python library is used to store the data on file.
- Storage Class: is used to wrap the Shelve python library.
- Challenges:
  - Debugging and testing - resetting the store data (highest score) during development and debugging. We solved this by adding a keyboard command (R key) to Reset the data.
  - Handling errors - when the application is run it reads the data, but the first time it runs it fails and raises an exception because the file is not there. We solved the problem by handling the exception and creating the file on first use.



Back End Design - Challenges

- View vs Model
  - it was very hard to maintain the separation between what code should be in the View vs in the Model. For example, keeping out anything relate to drawing the items on the screen from being part of the model. We achieved this by using a logical screen for the item.
- Data Permanence
  - it was difficult to find the right spots in the game cycle when to read and store the data to make sure it was not lost if the game was closed.
- Speed of the falling objects
  - Initially when working with screen coordinates from pygame a lot of magic numbers were used to control the speed based on trial and error. We switched to using a logical screen coordinate system based on percentages and that helped create a more predictable game experience.

Testing

- Acceptance Testing Procedure
  - Created a set of testing actions for all the supported functionality.
  - We updated the test as new functionality was added or modified.
  - If we found issues, we added tests to make sure the issue does not come back.
- Test runs
  - We routinely did short runs through the tests related to the functionality we were modifying.
  - Periodically did a full run through all the tests.
  - Added columns to the ATP spreadsheet to track if a given test was run so we could divy up the testing between members of the team.
- Usability testing
  - We let friends and family members try the game to see if they managed to use it and discover situations that we had not anticipated.

ATP

- The latest Acceptance Test Procedure can be found here
- Sound only works when run outside replit
- Testing Tips:
  - Use the S key to disable the background soundtrack music.
  - Use Q at any point to quit the app.
  - Use O to finish the game and jump directly to the Game Over screen.
  - Use R to reset the highest score.