Robotron: 2084 inspired Game

Written in PyGame with MVC architecture

John Montgomery

Supervisor: B. Harris

2020-2022

Abstract

In essence the project is an implementation of Robotron in PyGame, using Model-View-Controller, with a Flask based high scores bored. The main content of the code is in the game itself, with flask acting only as an API. This allows for shared usage of the route by a static web page, and by the PyGame code itself. The webpage is simply served off as static, where JS is able to communicate with the API to retrieve the information needed. The database used is Postgres.

Contents

1 Analysis **4**
1.1 What is MVC? 4
1.2 The Game 5
1.3 Limitations 5
1.4 Objectives 5
1.5 Design and Inspiration 6

2 Documented Design **8**
2.1 MVC in practice 8
2.2 Database 10
2.3 The API 11
2.4 The Server Setup 11
2.5 Security 11

3 Technical Solution **13**

4 Testing - TODO **14**

5 Evaluation - TODO **14**

6 Appendix **15**

List of Figures

1 A diagram showing the MVC architecture 4
2 Screen from original game - `https://arcadeblogger.com/2020/06/27/the-development-of-robotron/` 7
3 Advertising Material - `https://arcadeblogger.com/2020/06/27/the-development-of-robotron/` 7
4 Class diagram 8
5 Class diagram 9
6 Class diagram 9
7 Class diagram of characters 10
8 Flowchart of MVC 10
9 How tokens are generated 11

1 Analysis

1.1 What is MVC?

Model-View-Controller plays a large part in the project, the diagram [Figure 1] shows the main way that MVC works. It isolates the components of the game into 3 main components. The View, which is the screen, or what the user will see. The controller, which is where the user interacts with the game, in this case it is the interaction with the keyboard. The model, which is the part the user never interacts with, and stores the state of the game and current information about it.

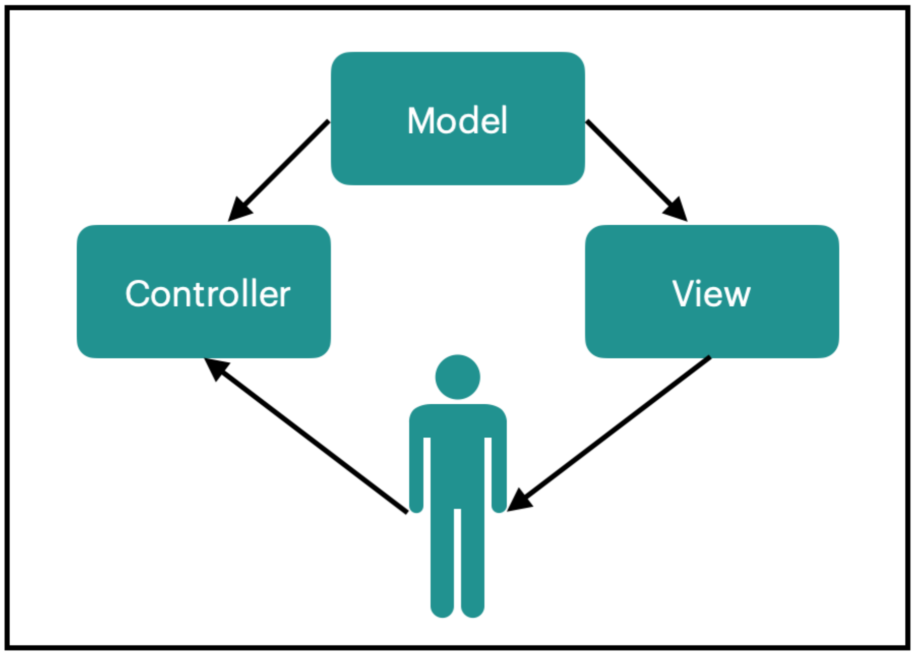


Figure 1: A diagram showing the MVC architecture

There are many benefits to this set up, for example, it will easily allow me to swap out what controller is used. If desired, it is much simpler to replace the keyboard as the human interface, and replace it with a game controller. Even more useful may be the ability to remove the controller and view entirely, allowing for a streamlined game which an AI could learn how to play. This flexibility, along with ease of programming is what drew me to use MVC for the game.

Another important information is the way information travels between the 3 sections. This is done with events, and an event manager is responsible for maintaining the sending and receiving of events through the system. A similarly important section is the States, and state machine, which controls the current ‘state’ the game is in, that is to say what level is being played, or what screens should be shown, such as a loading or help screen.

1.2 The Game

“Robotron: 2084” was released in 1982 by Williams Electronics. It was revolutionary as a dual stick shooter, was high energy and loved by many. This is important to capture into the game, where I want it to have a similar feeling to the original game, with some modern twists.

The game is about a species of ‘Robotrons’ created by humans in the year 2084, after realising their failings and created an advanced species. The goal is to save the humans (Mommies, Daddies and Mikeys), whilst fighting the robots, which have many kinds. The most basic are electrodes, which are static obstacles that kill on contact, but can be shot by players. The other basic enemy is the grunt, which is simply a basic soldier, which kills on contact, but moves towards the player. There are some other robots that will be talked about and implemented later, but the details about them are less important.

1.3 Limitations

The dual stick shooter nature means the player uses one joystick to move, and one joystick to shoot. This is difficult to implement well with a keyboard, but a simple setup which I am using is having WASD to move, and IJKL to shoot. Holding 2 keys diagonally at the same time will result it movement in an angle, allowing for shooting in 8 directions, and moving in 8 too.

Robotron is a fast fast game, I had to slow it down slightly in order to make it more playable on my laptop, and so it does feel somewhat different to the original. However by slowing it as I have I have made it a much smoother game to play.

1.4 Objectives

1. Create basic playing ability
   (a) Player can move in 8 directions
   (b) Player can shoot in 8 directions
   (c) Players animation is correct for direction of travel
2. Create basic enemies
   (a) Enemy is spawned in random position
   (b) Enemy can move
   (c) Enemy is animated
   (d) Enemy kills players
3. Create Loading Screens

    (a) Fuzzy loading screen
    (b) 'All test' screen
    (c) Home Screen

4. Create levels and transitions
    (a) Player moves between levels
    (b) Level transitions
    (c) Player is invincible on load
5. Create the API
6. Create login system
    (a) Basic API sign up works
    (b) GUI interactions with PyGame
7. High Scores
    (a) Top 10
    (b) Player Search
8. Create sounds with Game
9. Create scoring and score counter
10. Create a life counter
11. Automate testing on API and basic functions in PyGame

1.5 Design and Inspiration

The design for all the game is heavily taken from the original game. I used many places to research this, but below is a selection of screenshots and videos which were used in the creation of the game.

- https://www.youtube.com/watch?v=ccltMtkFBSI
- https://www.youtube.com/watch?v=aOVA2Axxfdk

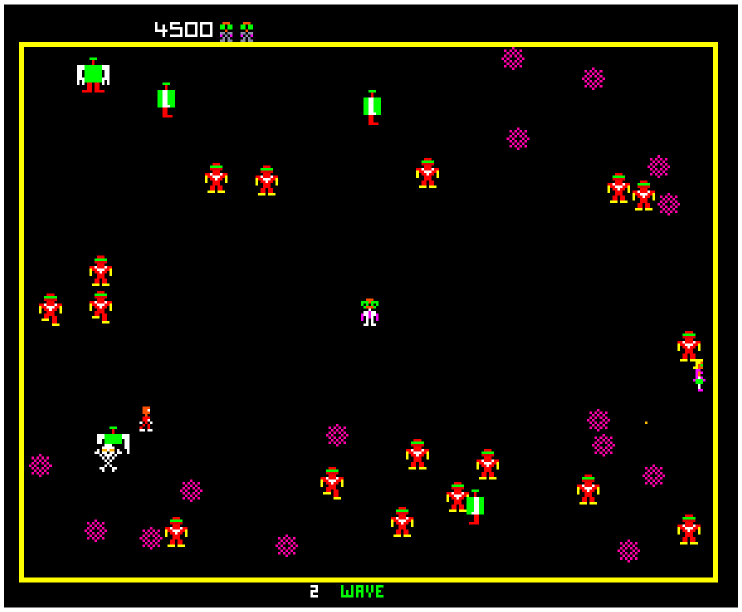
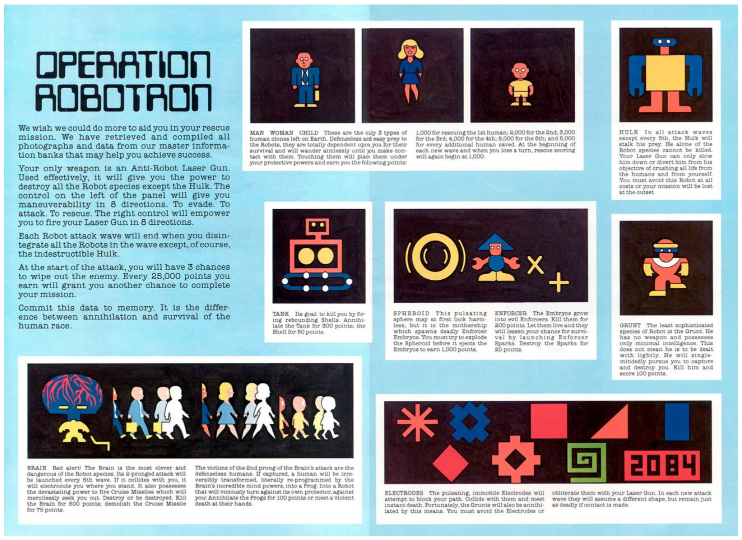


Figure 2: Screen from original game - `https://arcadeblogger.com/2020/06/27/the-development-of-robotron/`



OPERATION ROBOTRON

We wish we could do more to aid you in your rescue mission. We have retrieved and compiled all photographs and data from our master information banks that may help you achieve success.

Your only weapon is an Anti-Robot Laser Gun. Used effectively, it will give you the power to destroy all the Robot species except the Hulk. The control on the left of the panel will give you maneuverability in 8 directions. To evade. To attack. To rescue. The right control will empower you to fire your Laser Gun in 8 directions.

Each Robot attack wave will end when you disintegrate all the Robots in the wave except, of course, the indestructible Hulk.

At the start of the attack, you will have 3 chances to wipe out the enemy. Every 25,000 points you earn will grant you another chance to complete your mission.

Commit this data to memory. It is the difference between annihilation and survival of the human race.

MAN WOMAN CHILD These are the only 3 types of human clones left on Earth. Defenseless and easy prey to the Robots, they are totally dependent upon you for their survival and will wander aimlessly until you make contact with them. Touching them will place them under your protective powers and earn you the following points: 1,000 for rescuing the 1st human; 2,000 for the 2nd; 3,000 for the 3rd; 4,000 for the 4th; 5,000 for the 5th; and 5,000 for every additional human saved. At the beginning of each new wave and when you lose a turn, rescue scoring will again begin at 1,000.

HULK In all attack waves except every 5th, the Hulk will stalk his prey. He alone of the Robot species cannot be killed. Your Laser Gun can only slow him down or divert him from his objective of crushing all life from the humans and from yourself. You must avoid this Robot at all costs or your mission will be lost at the outset.

TANK Its goal: to kill you by firing rebounding Shells. Annihilate the Tank for 300 points; the Shell for 50 points.

SPHEROID This pulsating sphere may at first look harmless, but it is the mothership which spawns deadly Enforcer Embryos. You must try to explode the Spheroid before it ejects the Embryos to earn 1,000 points.

ENFORCER The Embryos grow into evil Enforcers. Kill them for 200 points. Let them live and they will lessen your chance for survival by launching Enforcer Sparks. Destroy the Sparks for 25 points.

GRUNT The least sophisticated species of Robot is the Grunt. He has no weapon and possesses only minimal intelligence. This does not mean he is to be dealt with lightly. He will single-mindedly pursue you to capture and destroy you. Kill him and score 100 points.

BRAIN Red alert! The Brain is the most clever and dangerous of the Robot species. Its 2-pronged attack will be launched every 5th wave. If it collides with you, it will electrocute you where you stand. It also possesses the devastating power to fire Cruise Missiles which will mercilessly seek you out. Destroy or be destroyed. Kill the Brain for 500 points; demolish the Cruise Missile for 75 points. The victims of the 2nd prong of the Brain's attack are the defenseless humans. If captured, a human will be irreversibly transformed, literally re-programmed by the Brain's incredible mind powers, into a Prog. Into a Robot that will viciously turn against its own protector, against you! Annihilate the Progs for 100 points or meet a violent death at their hands.

ELECTRODES The pulsating, immobile Electrodes will attempt to block your path. Collide with them and meet instant death. Fortunately, the Grunts will also be annihilated by this means. You must avoid the Electrodes or obliterate them with your Laser Gun. In each new attack wave they will assume a different shape, but remain just as deadly if contact is made.

Figure 3: Advertising Material - `https://arcadeblogger.com/2020/06/27/the-development-of-robotron/`

2 Documented Design

The main design aspect is the MVC architecture and how it forms the basis of the game. Fig 1, from the analysis section, gave a very brief, high level and non technical view of MVC. In this section I will go into more detail about my own implementation, and how it works in greater detail. This section also details the database on the web side, the API, the technical setup of the servers, the data structures and HCI designs.

2.1 MVC in practice

In the analysis section I gave a very high level overview of MVC, this part will detail further into my design on its implementation in python. The first main, basic components of MVC are of course, the model, the view, and the controller. Figure 6 shows the 3 classes diagrams for each of the implementations of these in python.

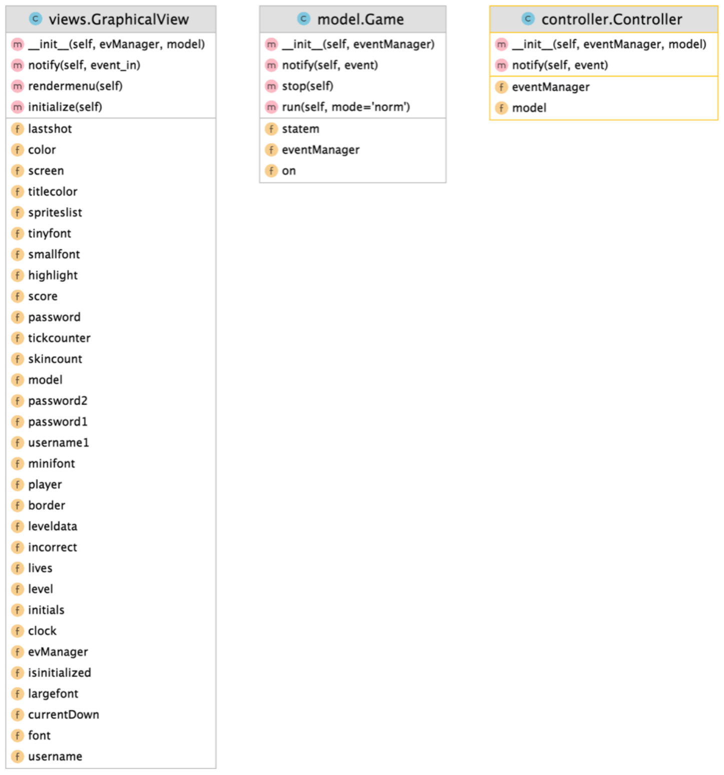


Figure 4: Class diagram

On top of these key features, there's also a range of other important cogs in the system. One of the most important, to allow for the communication between the M, V and C are Events, and an event manager. A Sample of events, and the event manager is given in Fig fig:events.

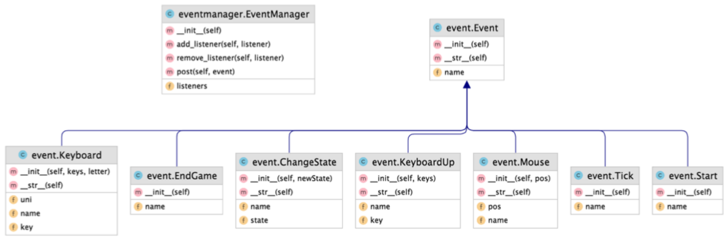


Figure 5: Class diagram

The other key class is the state machine. Each state is not given its own class, rather there is a constant number which is attributed to a given state. The states are used for the larger changes in the program and events are for the smaller interactions, and ticks.

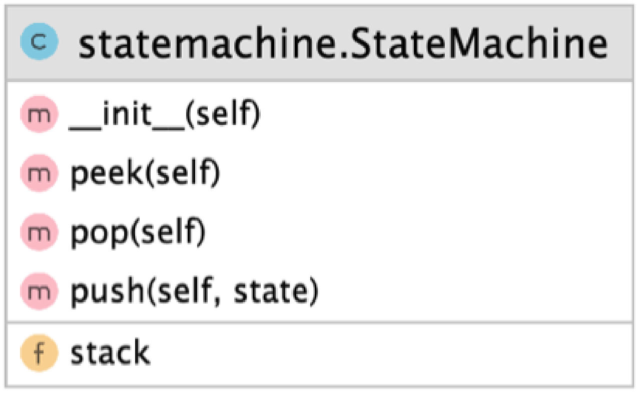
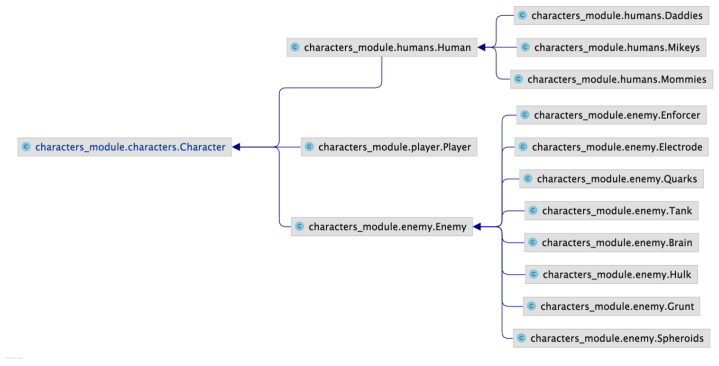


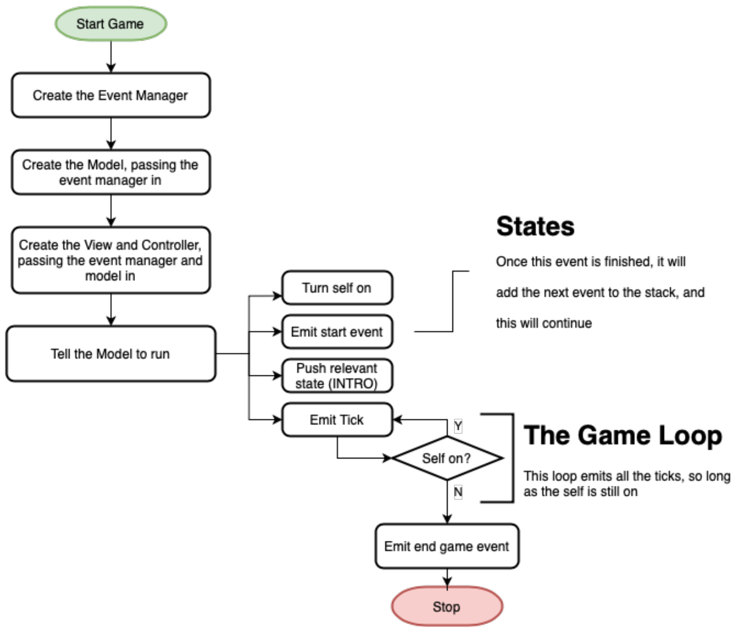
Figure 6: Class diagram

In order to run through a basic idea of what happens when the program is run, I have created a step by step flowchart. This flowchart [Fig 9] is a gross oversimplification, but works as a high level description of what it is my code is doing when executed.



characters_module.characters.Character
characters_module.humans.Human
characters_module.player.Player
characters_module.enemy.Enemy
characters_module.humans.Daddies
characters_module.humans.Mikeys
characters_module.humans.Mommies
characters_module.enemy.Enforcer
characters_module.enemy.Electrode
characters_module.enemy.Quarks
characters_module.enemy.Tank
characters_module.enemy.Brain
characters_module.enemy.Hulk
characters_module.enemy.Grunt
characters_module.enemy.Spheroids

Figure 7: Class diagram of characters



Start Game
Create the Event Manager
Create the Model, passing the event manager in
Create the View and Controller, passing the event manager and model in
Tell the Model to run
Turn self on
Emit start event
Push relevant state (INTRO)
Emit Tick
Self on?
Y
N
Emit end game event
Stop
States
Once this event is finished, it will add the next event to the stack, and this will continue
The Game Loop
This loop emits all the ticks, so long as the self is still on

Figure 8: Flowchart of MVC

2.2 Database

This section will show the database design and set up, and explain some of the SQL used in the program. Fig 10 shows the database diagram.

[TODO - Database diagram]

There are 3 tables, scores, users and tokens. The scores database has 2 fields which store the users ID and their Score for a given game. The Users table stores the users info, such as emails, password hashes, etc, and then the tokens database is used to store validated tokens (with time limits) which are used to validate the GUI and avoids needing to login to the the program every time the game is run. Fig 11 shows the process of creating the tokens.

2.3 The API

The leaderboard contains only 6 routes, as these were all that are necessary, the details for the routes are detailed in the table below.

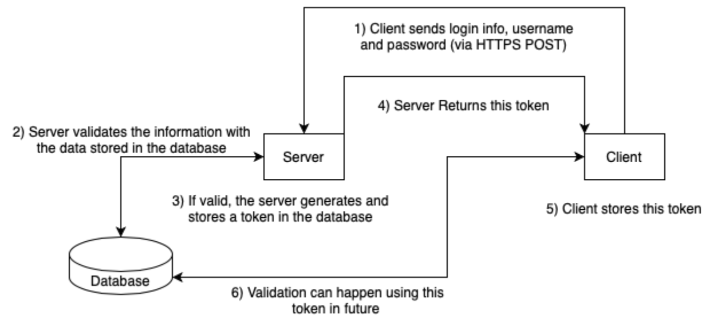

1) Client sends login info, username and password (via HTTPS POST)
4) Server Returns this token
2) Server validates the information with the data stored in the database
Server
Client
3) If valid, the server generates and stores a token in the database
5) Client stores this token
Database
6) Validation can happen using this token in future

Figure 9: How tokens are generated

ROUTE	METHOD	DESCRIPTION
/leaderboard	GET	Returns JSON of top 10 users (initials + scores) in Database
/user/userid	GET	Returns JSON of top score
/username/userid	GET	Returns ID of given username
/login	POST	Logs in a user, sends token, or logs user in with token
/addscore	POST	Adds a score, given score and a token
/adduser	POST	Adds a user to the database

2.4 The Server Setup

Fig 12 shows the set up the server is in. All using AWS, there is an RDS Postgres database, and EC2 instance (this is the server running the actual flask) and then an S3 bucket to handle sending the static files. It may also be possible to use NGINX or Apache to serve and handle the API. This system may end up being better, so my current architecture could change.

2.5 Security

Because the database and client handles personal details like email and passwords, there needs to be a thought to security. First off, there is an enforcement of passwords and a strong policy. Users passwords will need to be 8 characters, with 1 special, and my plan is to check them against a list of common passwords (rocky.txt) using hashes. For this I will probably use MD5, or something even faster. However it is important to avoid these fast algorithms when hashing passwords for storage. As such, passwords will undergo key derivation through bcrypt, an algorithm which not only salts, but performs many rounds of hashing. I could implement a similar algorithm using the basic functions like SHA, but rolling your own crypto is never good, so its going to be done with bcrypt, as this is essentially the best option available, and more than secure enough.

To help further security, HTTPS is being used for all the sending and receiving of data, this avoids man in the middle attacks of the data as it gets sent over the internet.

3 Technical Solution

Im working on a way to easily do this. I am using LaTeX as this will allow me to not have to change the code on my report manually, as it will link to the files. However i am still learning. This file will also update on my github, so if you go to https://github.com/john-montgomery2003/Robotron2084 you will be able to find a pdf of this document which is more up to date.

4 Testing - TODO

5 Evaluation - TODO

6 Appendix

Name	**Server/Web/Game/Dev**	**Use**
Flask	Server	Handles the API and web on server side
SQLalchemy	Server	Used to connect to the Postgres database
BCrypt	Server	Key derivation
Waitress	Server	WSGI server
PyGame	Game	Graphics and input handling
S3	Server	AWS static file hosting / serving
EC2	Server	AWS server to run flask app
Hetzner	Server	Alternative option to run flask and serve files
PyCharm	Dev	My IDE choice