|  |  |
| --- | --- |
| **COMP704 – Machine Learning** | **Worksheet WK 2** |
| **Architecting Games, HTTP Serving and Regression** | |

**Introduction**

In the lecture this week, we looked at regression as a form of machine learning. This workshop will cover that, but I also want to look at how to build simple games in a design-focused way and have a proper look at working with HTTP for data collection.

**Architecting Games in a structured way**

Universal Modelling Language (UML) is often cited as the ‘professional’ way to define software architecture and there’s a lot of truth in that statement. However, it’s not the only way to understand and define software architecture and, indeed, it gives us a fairly open format of ‘lines and boxes’ to describe systems through hierarchy and flow.

Video games often suffers with UML as the general starting point for UML is use-case collaborations, something that game designers are not that good at delivering:

|  |
| --- |
| Image result for use case collaboration |

Instead, game designers tend to be better at producing design documents, however, this creates a challenge for programmers to work with as they often tend to be verbose and full of contradictions, something that use-case collaborations can avoid.

Booch gives us an alternative to this situation with linguistic decomposition, where nouns, verbs and adjectives can be taken from a design document to give us a rough idea as to what classes, functions and data is likely to look like:

|  |
| --- |
|  |

This will give us a simple class pass:

A screen shot of a social media post

Description automatically generated

That we can use as a base point for filling in the ‘gaps’ in our technical design

A screenshot of a cell phone

Description automatically generated

**Build your own class hierarchy**

Here’s a typical designer take on a game design, use the Booch approach above to work out how the game architecture could look and then refine it to make it more workable

|  |
| --- |
| The objective of Asteroids is to score as many points as possible by destroying. The player controls a triangular-shaped ship that can rotate left and right, fire shots straight forward, and thrust forward. As the ship moves, momentum is not conserved — the ship eventually comes to a stop again when not thrusting. The player can also send their ship into hyperspace, causing it to disappear and reappear in a random location on the screen (with the risk of self-destructing or appearing on top of an asteroid).  Each stage starts with a few asteroids drifting in random directions on the screen. Objects wrap around screen edges — for instance, an asteroid that drifts off the top edge of the screen reappears at the bottom and continues moving in the same direction. As the player shoots asteroids, they break into smaller asteroids that frequently move faster and are more difficult to hit. Smaller asteroids also score higher points. The player has three lives, whenever the player is hit by an asteroid the player loses a life. After dying, the player will be re-positioned back in their start position on the screen - once it is safe to do so, i.e. so the player won't get killed as soon as the game continues. When the player has lost all their lives a game over screen will be displayed and the player will be taken back to the game wrapper.  When the game is loaded, the player is initially presented with a game wrapper inviting the player to press the space bar to start the game. On pressing the space bar the game will start. During game play, the player may press the escape key to pause the game play, this will present that player with an in-game menu, containing the options resume or quit. Pressing escape on the resume option will return the player to the game, i.e. removing the pause. Pressing escape on the quit option will return the player to the game wrapper. Pressing the escape key on the game wrapper will quit the game. |

**Function and form in game technical design**

**HTTP Serving**

**Regression**