FALMOUTH UNIVERSITY

Creating Technical Posters

COMP2x0: Individual Specialist Computing Project
BA Game Development
BSc Computing for Games

- Creating Technical Posters
  - Tends to be a bit of a challenge for students
    - Across all 3 years of the degree programme

- Creating Technical Posters
  - Tends to be a bit of a challenge for students
    - Across all 3 years of the degree programme

    - Have a look at previous student feedback to see ‘issues’
      - *contains a flowchart describing behaviour and a hard to read blueprint screenshot*
      - *It features several blueprints but would benefit from some UML flowcharts / state diagrams to describe the implemented behaviours*
      - *consists largely of hard to read blueprints with little explanation*
      - *contained little detail and some hard to read screenshots of code*
      - *almost completely devoid of any technical information*
      - *It was difficult to see what the poster was trying to explain*

- Creating Technical Posters
  - Tends to be a bit of a challenge for students
    - Across all 3 years of the degree programme
  - Scope to vacuum up marks on your degree
    - Technical posters carry significantly more marks / effort than anything else does
    - Potential to shift your degree up by a classification by doing ‘good’ posters
    - Don’t leave them ‘til the last minute
      - I heard a few horror stories about people doing them on the morning of presentation
    - The law of diminishing returns is your friend

- Creating Technical Posters
  - Technical posters, why?
    - We’re not going to make them for interviews, so why bother getting good at them
    - Technical posters are a vehicle for you to develop your skills in articulating technical problems and solutions using ‘industry standard’ terms, tools and techniques.
    - How technical posters help in interviews
      - You’ll be asked to talk about problems you’ve solved and how you solved them
      - You need to be able to articulate that into UML, design patterns, pseudo code & process
      - You may even end up drawing things on a whiteboard

- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
      - **When** do you do them
      - **Where** do you do them
      - **Why** do you do them
      - **How** do you do them

- Creating Technical Posters
  - Making good posters
    - Let's think about this from 5Ws&H perspective
      - **Who** are they for
        - » **Lecturing staff** to assess your ability to articulate technical problem solving
        - » **You** to develop your ability to articulate technical problem solving
      - **What** is their purpose
      - **When** do you do them
      - **Where** do you do them
      - **Why** do you do them
      - **How** do you do them

- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
        - » Demonstrate you ability to solve interesting technical problems
        - » A vehicle for you to articulate how you solve interesting technical problems
      - **When** do you do them
      - **Where** do you do them
      - **Why** do you do them
      - **How** do you do them

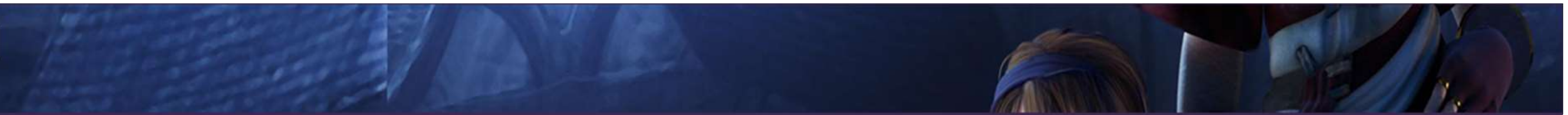
- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
      - **When** do you do them
        - After you have solved interesting technical problems
        - Preferably not just before their presentation
      - **Where** do you do them
      - **Why** do you do them
      - **How** do you do them

- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
      - **When** do you do them
      - **Where** do you do them
        - Present in an environment where there are lots of presentations to be done
          - Get interest quickly
          - Show clear ideas and a clear process of problem solving
      - **Why** do you do them
      - **How** do you do them

- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
      - **When** do you do them
      - **Where** do you do them
      - **Why** do you do them
        - » Within the course: to develop your skills at presenting problem solving
        - » Outside of the course: to demonstrate interesting solutions to challenging problems
      - **How** do you do them

- Creating Technical Posters
  - Making good posters
    - Let’s think about this from 5Ws&H perspective
      - **Who** are they for
      - **What** is their purpose
      - **When** do you do them
      - **Where** do you do them
      - **Why** do you do them
      - **How** do you do them
        - » ...

- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - A problem that needed to be addressed
      - The solution that worked
      - The outcome and its impact



- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - A problem that needed to be addressed
      - The solution that worked
      - The outcome and its impact
    - It needs to ‘fit’ on a reasonable sized piece of paper
      - Can’t be a really big problem
      - Or a really complex solution
      - Or an outcome with lots of caveats

- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - A problem that needed to be addressed
      - The solution that worked
      - The outcome and its impact
    - It needs to 'fit' on a reasonable sized piece of paper
      - Can't be a really big problem
      - Or a really complex solution
      - Or an outcome with lots of caveats

    → A Goldilocks problem
      - Not too big, not too small, not too complex & not too trivial

- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - A problem that needed to be addressed
        - » We know there are lots of problems that need to be addressed in game development, some are more interesting to present than others
        - » For a poster,
          - do you have problems that will make people happy when they are fixed?
          - do you have problems that required ‘interesting’ technical solutions?
          - Etc
        - » Can you articulate it in not very much writing?
      - Ultimately:
        - » What is the problem
        - » Why is it an issue

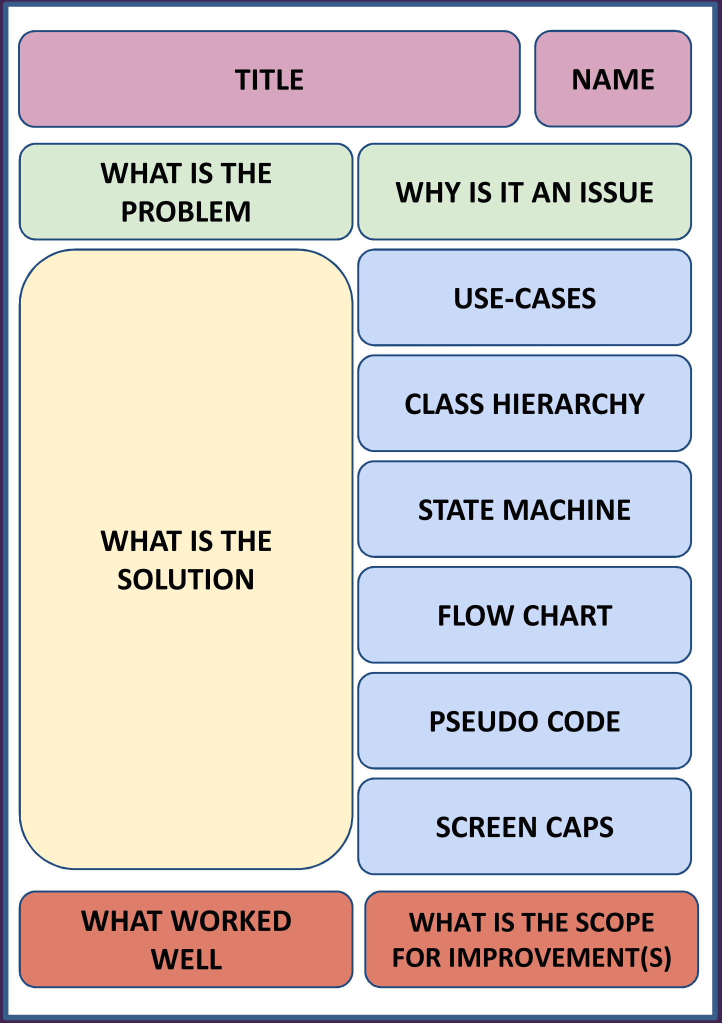
- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - The solution that worked
        - » Writing your poster towards the end of development (not the morning of the presentation), you will have a good handle on:
          - What ultimately worked
          - The steps you had to go through
          - The dead ends and blind alleys you encountered
          - The key features of you solution's form & function, and data model
      - Ultimately:
        - » This is about articulating detail
          - Enough detail to express what's important
          - Ignore what's not important

- Creating Technical Posters
  - Making good posters
    - The heart of a good poster is a good story
      - The outcome and its impact
        - Hopefully, your solution worked and feature ‘X’ was added to the game
        - However, does your solution provide impact beyond the scope of adding a feature to a game or removing a work item from a backlog.
      - Ultimately:
        - This is about closing the story
          - Did your solution meet its goals
          - Did it add value
          - Does it open up new areas of interest / approaches
          - Does it have a positive impact for co-workers

FALMOUTH UNIVERSITY

- Creating Technical Posters
  - Making good posters

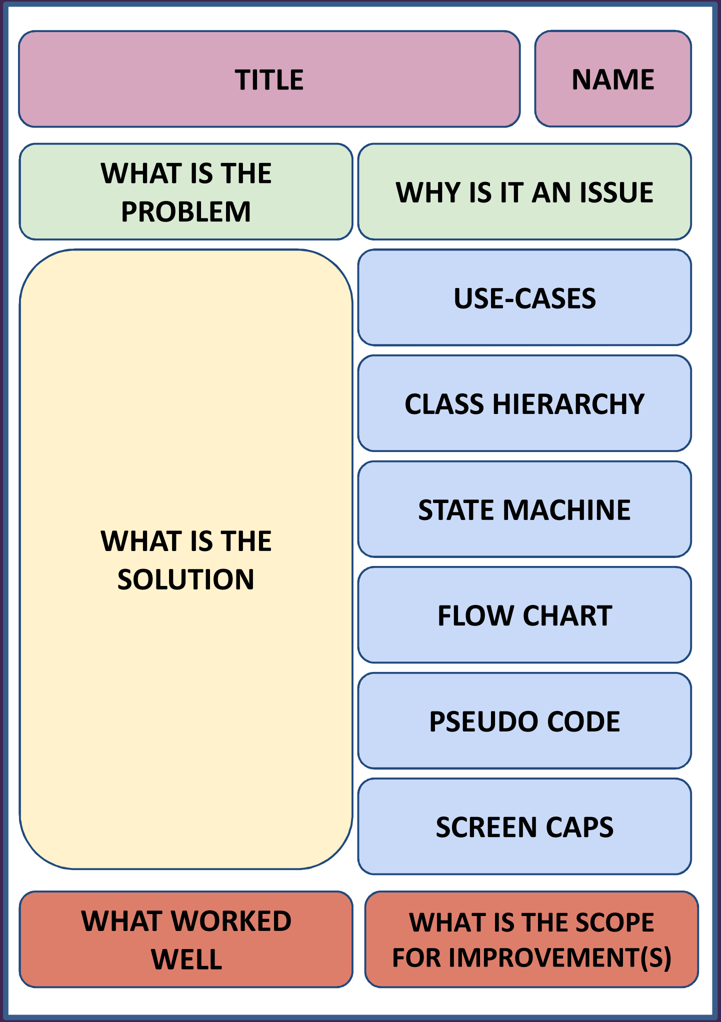
- Here’s my conceptual model of good posters
  - There’s five parts:
    - Name & Title
    - Problem definition
    - Solution description
    - Modelling tools
    - Outcome and impact
  - It’s conceptual, so posters don’t have to look like this, but they should use the parts
    - Apart from all the modelling tools, just use what makes sense for your poster. Don’t use all of them



FALMOUTH UNIVERSITY

- Creating Technical Posters
  - Making good posters

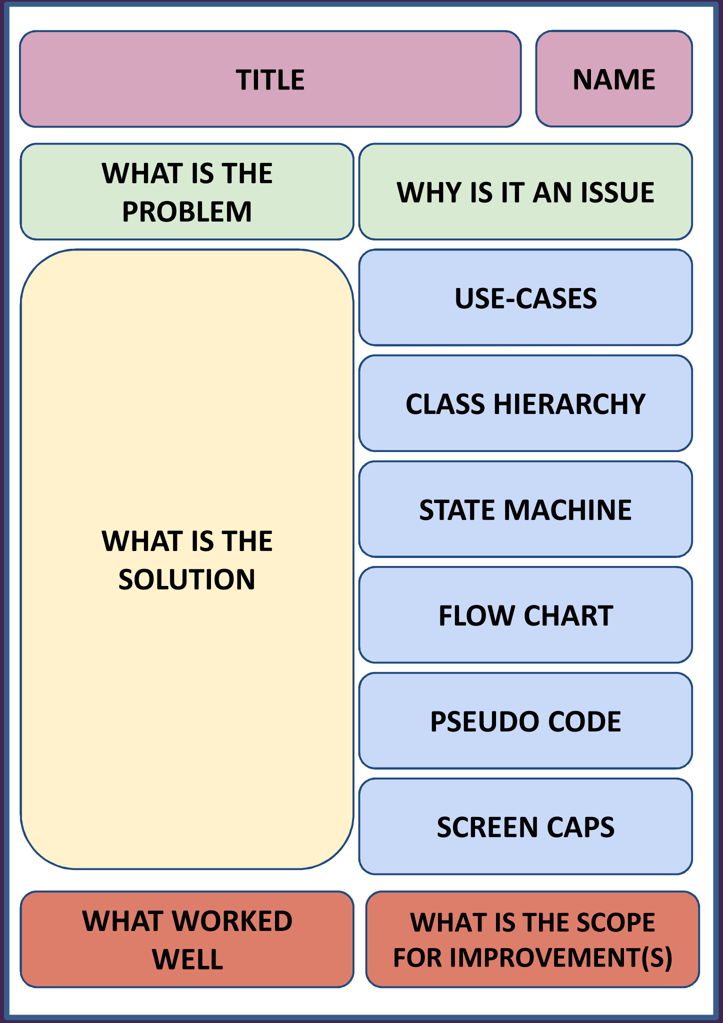
- Name & Title
  - The poster should have a title that accurately reflects the content of the poster
  - Avoid puns and oblique references
  - Include your name



FALMOUTH UNIVERSITY

- Creating Technical Posters
  - Making good posters

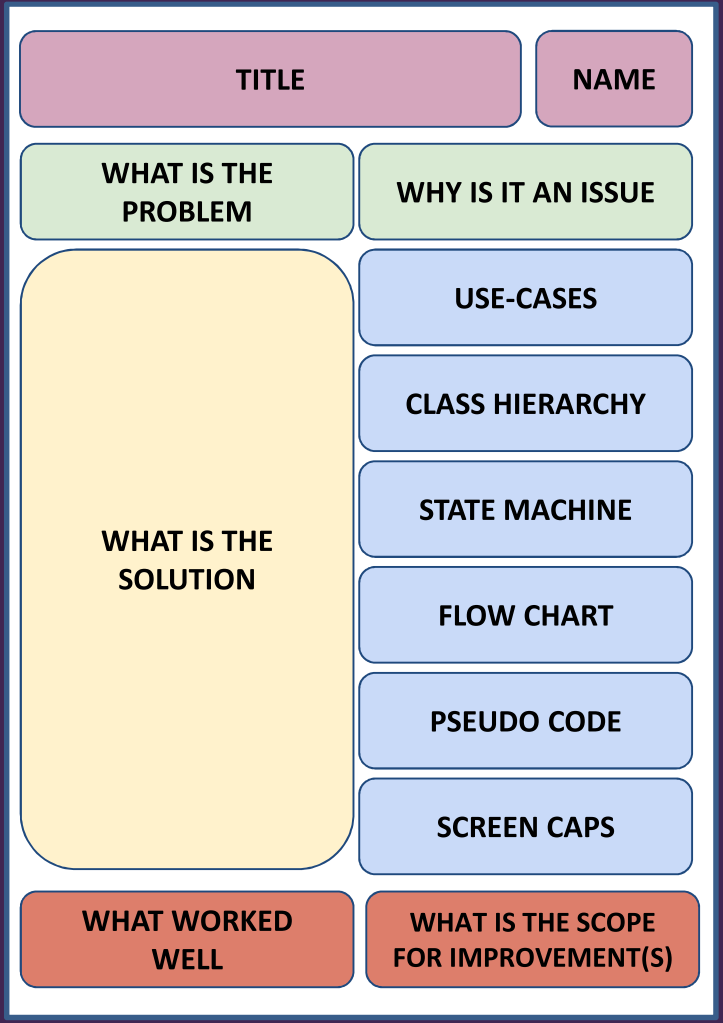
- Problem Definition
  - This needs to detail what the actual problem is that you are addressing and why it is a problem / issue
  - Depending on the nature of the problem, it may make sense to articulate it with:
    - A screenshot if you are talking about visual improvements
    - A use-case if you are talking about people focused improvements / issues
    - A class diagram if you are talking about refactoring structure
    - etc



FALMOUTH UNIVERSITY

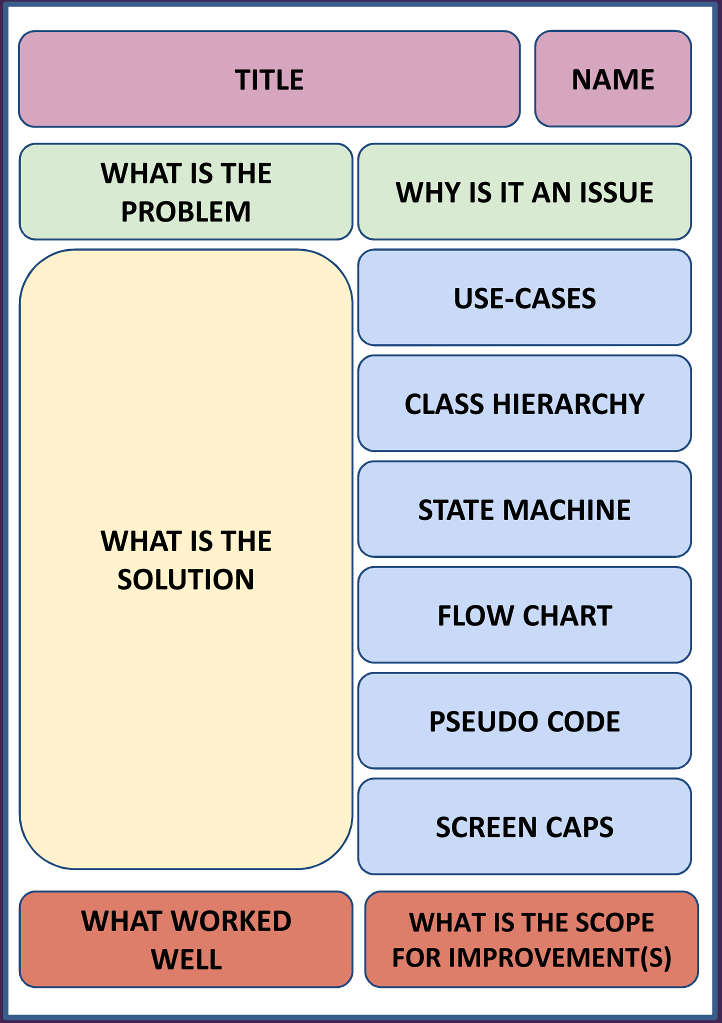
- Creating Technical Posters
  - Making good posters

- Solution description
  - This needs to detail what the solution is and how it is implemented
  - This should contain some text with references to whatever modelling tools make sense for your description
  - Bear in mind, that your audience will not want to engage with a ‘wall of text’ and a picture is worth a thousand words (as long as it adds value), so you need to carefully balance your writing against your modelling tools.
  - Ideally, your text will reference your modelling tools with the detail in the model rather than the text



- Creating Technical Posters
  - Making good posters

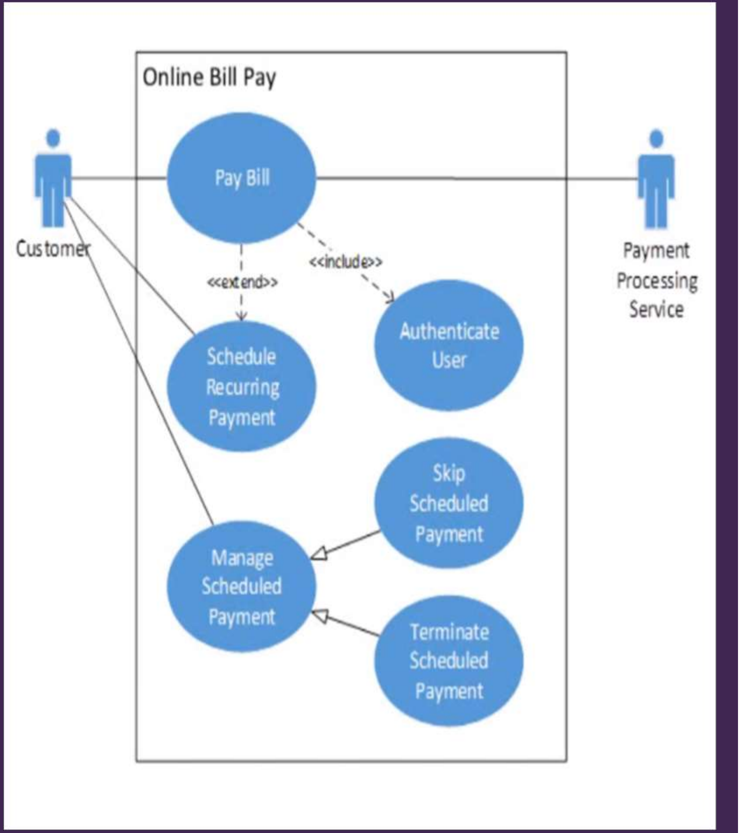
- Modelling Tools
  - These are a selection visual tools for articulating your solution and draw from UML and other modelling approaches and methods of presenting software architecture and function
  - Do not attempt to use all of them for your poster – it will create an awful mess
  - Feel free to remove content from your models so that you can easily express what you want for your poster -> this is not a technical plan, the models are just illustrative.
  - DO NOT USE BLUEPRINT SCREENSHOTS



FALMOUTH UNIVERSITY

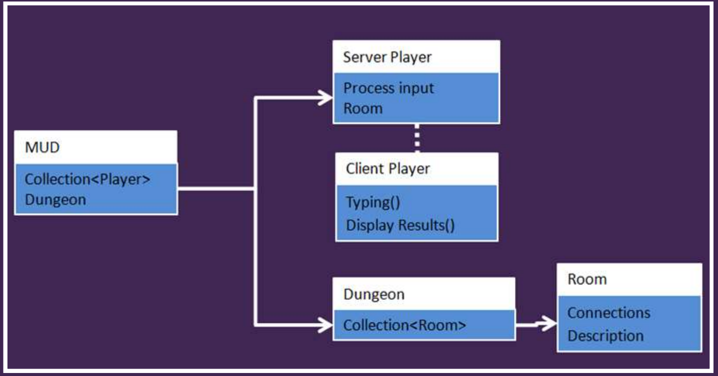
- Creating Technical Posters
  - Making good posters

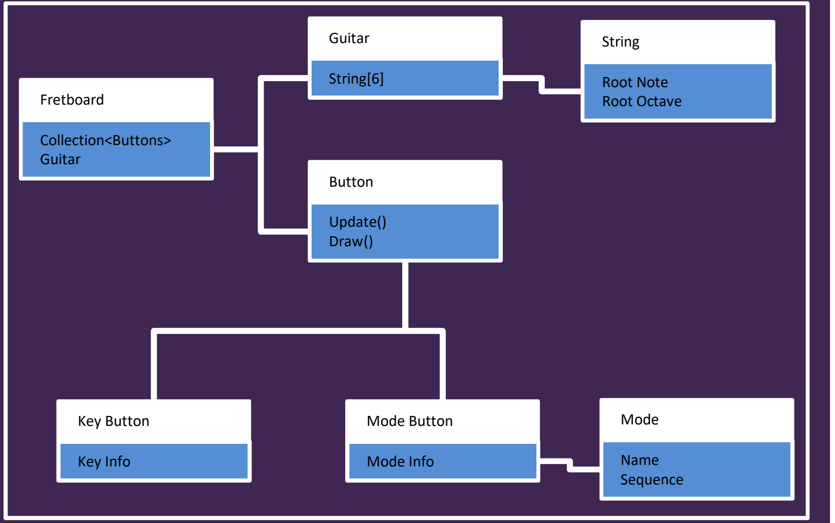
- Modelling Tools
  - Use-cases
    - Use these to model human-centric processes
    - Particularly useful for:
      - Showing people-centric processes (art / production / build / QA) pipelines
    - Watch out for:
      - Detail
      - Legibility of text / colour combinations



- Creating Technical Posters
  - Making good posters

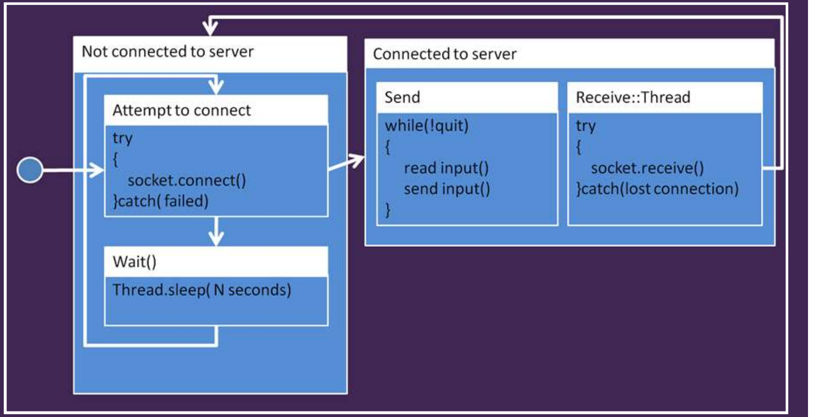
- Modelling Tools
  - Class hierarchy / class diagram
    - Use this to show the relationship between classes
    - Particularly for inheritance & abstraction
      - i.e. A base class defines certain interfaces & containers
    - Also to show class composition
      - i.e. Class A has objects of type B & C in it

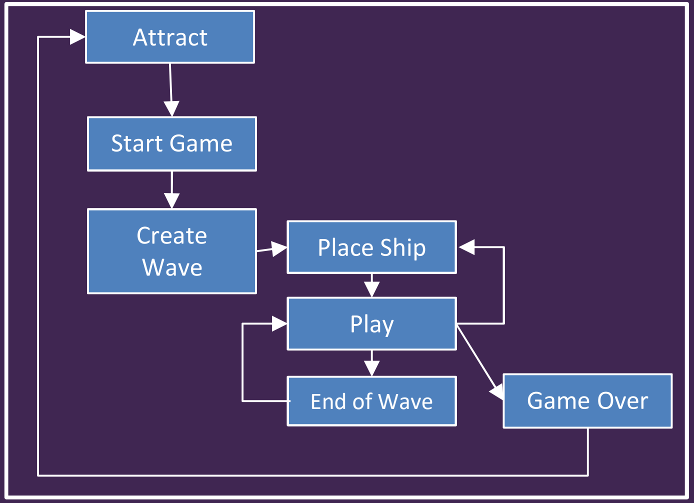




- Creating Technical Posters
  - Making good posters

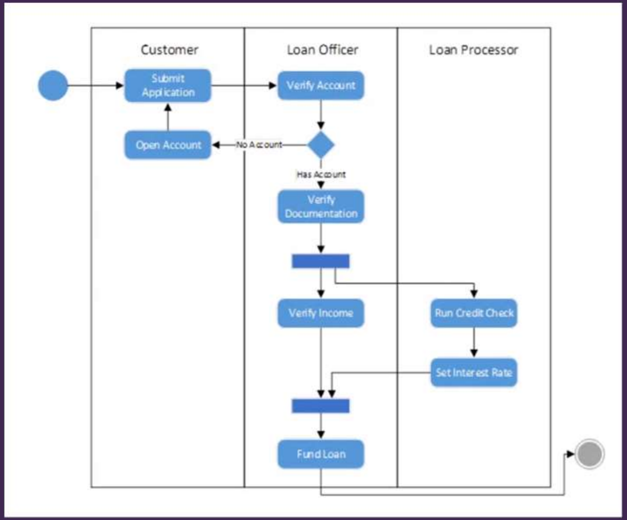
- Modelling Tools
  - State Machine / State Diagram
    - Use this to show the high-level states within an application
      - What functions a system will implement
      - How control flow moves between states
    - Particularly useful for:
      - Game states
      - Baddie / AI states
      - UI flow





- Creating Technical Posters
  - Making good posters

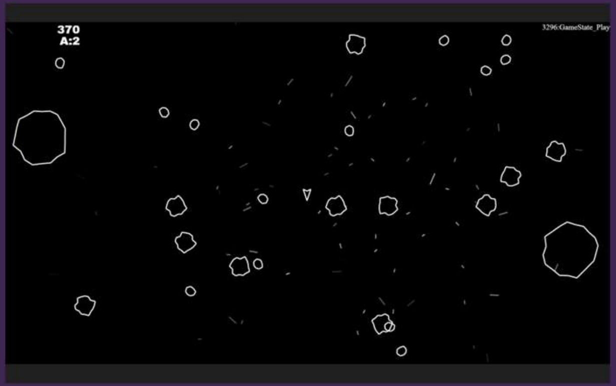
- Modelling Tools
  - Flow chart / Activity Diagram
    - Use this to show fairly medium-level functionality within states or functions
    - Particularly useful for:
      - Showing control flow between objects / systems
      - Functionality in functions / procedures without requiring the detail of pseudo code

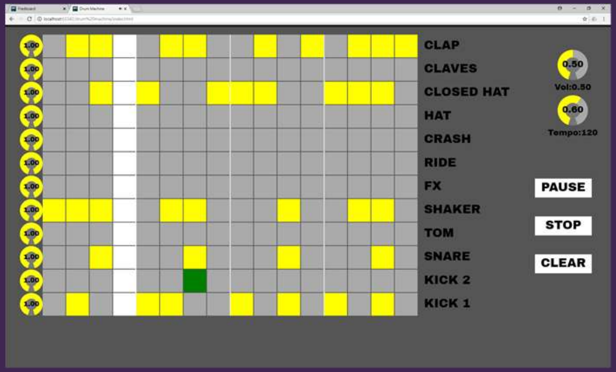


- Creating Technical Posters
  - Making good posters
- Modelling Tools
  - Pseudo code
    - Use this to show low-level functionality within functions / classes
    - Particularly useful for:
      - Functionality in functions / procedures
    - Watch out for:
      - Detail
      - Legibility of text / colour combinations
      - Visual Studio error marking ;)

```
generateValidNotes(mode, key)
{
   var valid_notes = [];

   var current_note = key;

   valid_notes.add(current_note );

   foreach(var step in mode)
   {
      current_note += step;

      valid_notes.add(current_note);
   }

   return valid_notes;
}
```

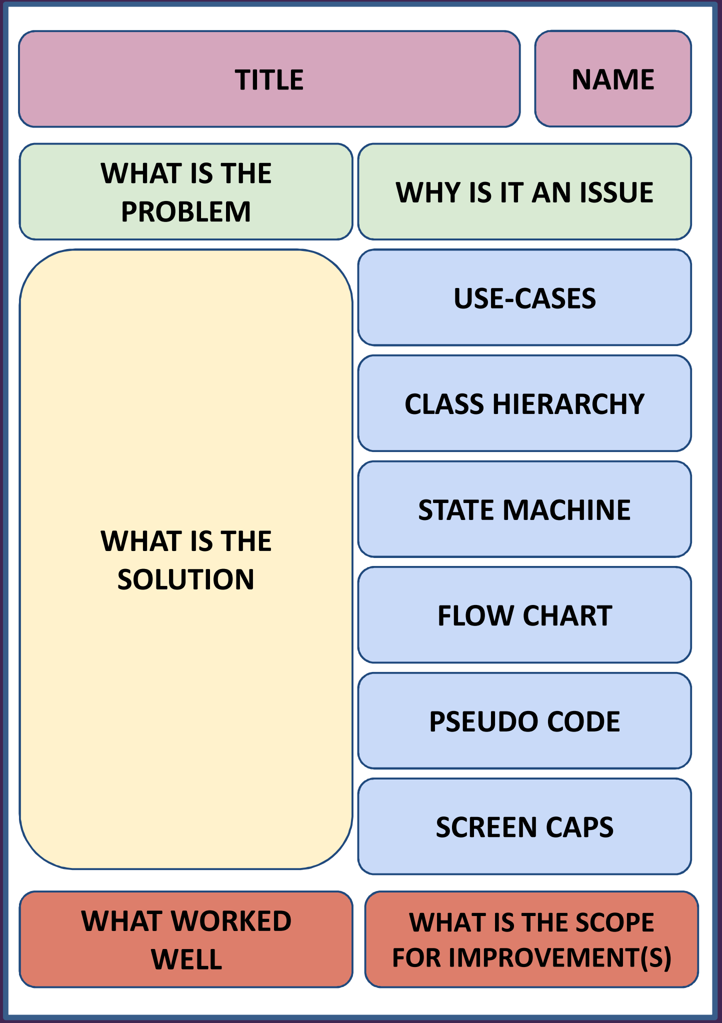
- Creating Technical Posters
  - Making good posters
- Modelling Tools
  - Screen caps
    - Use this to show the visual output of your game / application
    - Particularly useful:
      - It would take a lot of words to describe something visual
      - Show what your application looks like
    - Watch out for:
      - Loss of detail when shrunk to fit on poster
      - Darkness in printing process





- Creating Technical Posters
  – Making good posters

- Outcome & Impact
  – This section is to let the reader know what the outcome of your poster was:
    – Did it have a positive impact on the project
    – Has it opened new approaches
    – Etc
  – Being reflective and even self-critical are good



- Creating Technical Posters
  - Making good posters
    - An example
      - The core of the JS game engine is a state machine
        - » It has a virtual base class of state that provides init(), update() & draw() functions that each game state derives off
        - » State switching is done through statemachine.setState()
        - » Can be used for other state-based things that require update() / draw() split
      - Why do this?
        - » Relatively constrained issue to look at
        - » Shows the need for structural organisation in games

- Creating Technical Posters
  - Making good posters
    - An example
      - Problem definition
        - » Looking for a solution to manage a JS application as a state of states that can be transitioned between
        - » Why is this an issue
          - Software engineering says that divide and conquer is good for solving problems
          - Games naturally lend themselves to state-based structure
          - Using divide & conquer will be good for solving programming issues in a modular manner, particularly for testing
      - Solution description
        - » Statemachine class to store all the game states & handle functionality
          - Dictionary to map state names to state functions & data
          - Update and draw functions to deal with separating functionality
          - setState() &Update() functions to manage state switching

- Creating Technical Posters
  - Making good posters
    - An example
      - Modelling tools
        - Class diagram – show the inheritance
        - Pseudo code – show how state switching works
        - State diagram – show how a game is composed of states
      - Outcome and impact
        - This approach is good for games (lots of states)
        - Not so good for applications (one state)
        - Basic approach can be useful in other areas, though state machine methods may change

FALMOUTH UNIVERSITY

STATE MACHINES FOR GAMEFLOW IN JAVASCRIPT

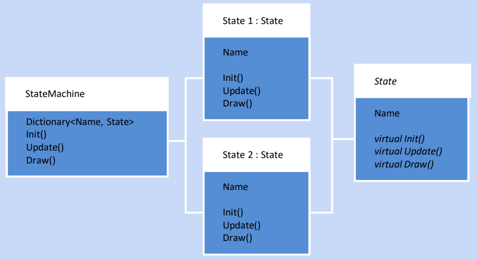
GARETH LEWIS

Introduction

JS and Canvas provides a light-weight development environment for creating games, but feature little functionality for managing game states within the context of a game. Hard-coding states through `case` or `if` statements is both time consuming and open to programming errors. A robust and re-usable game state framework would potentially offer a lot of advantages to developing JS games

Solution

My need for a robust game state machine came from developing an Asteroids clone for Canvas. Asteroids has a fairly simple set of game states [see state diagram below] and transitions between states which occur on user input, e.g. going from 'attract' mode to playing the game, or on game events, e.g. player getting killed or clearing a wave of asteroids.



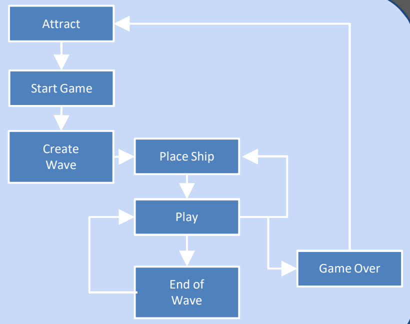
Class hierarchy of game state machine

```
class StateMachine
{
  Dictionary<String,State> stateLookup;
  String currentState = "";
  String pendingState = "";

  setState(state){pendingState = state;}

  update()
  {
     if(pendingState !=== '')
     {
       stateLookup[currentState].exit();
       currentState = pendingState
       stateLookup[currentState].init();
       pendingState = '';
     }
     stateLookup[currentState].update();
```

State-switching pseudo code

The structural solution to was to create an abstract game state class and derive all the actual game states from it [see class hierarchy above]. These states were stored in the state machine class as a dictionary of state names to base states. The state machine was made part of a `Game` singleton class giving the entire game code access to the state machine to allow global state changing from any point within the codebase, particularly from within a state's `update()` function as this was where most state changes originated.

Runtime state management and state changing occurred through the interaction of the `setState()` and `update()` functions in the State Machine class. The state switching pseudo code [above] details a lightweight `setState()` function that just records a state change request. This used by the `update()` function to determine if a state change is required. If so, the current state is exited, the state name is changed and the new state runs its init function before returning to the standard path, which will then call the state's update().



State diagram of Asteroids game

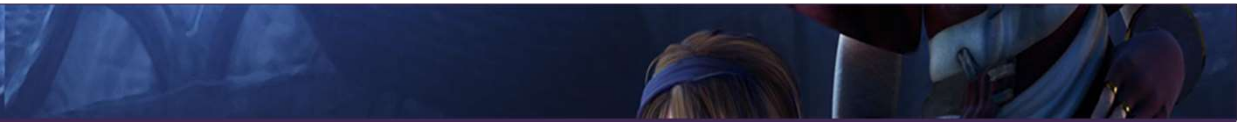
Conclusions

This approach has worked successfully on the Asteroids game I developed in JS/Canvas. Breaking the game down into a set of states has allowed me to keep a close relationship between game design, as the state diagram, and the code that was implemented from those states.

State transitions have been made fairly trivial as just a function that is called with a desired state label.

The one downside I discovered through testing and debugging was that if multiple state change requests occurred in an update, only the last request would be acted on, though this has currently been addressed with some questionable code, I need to address this with a better solution.

This approach has shown me that re-usable state management could be applied in other game programming areas and I'm keen to look at baddie AI and UI work in the next instance.



- Questions