LUMENS

PROJECT REPORT 2

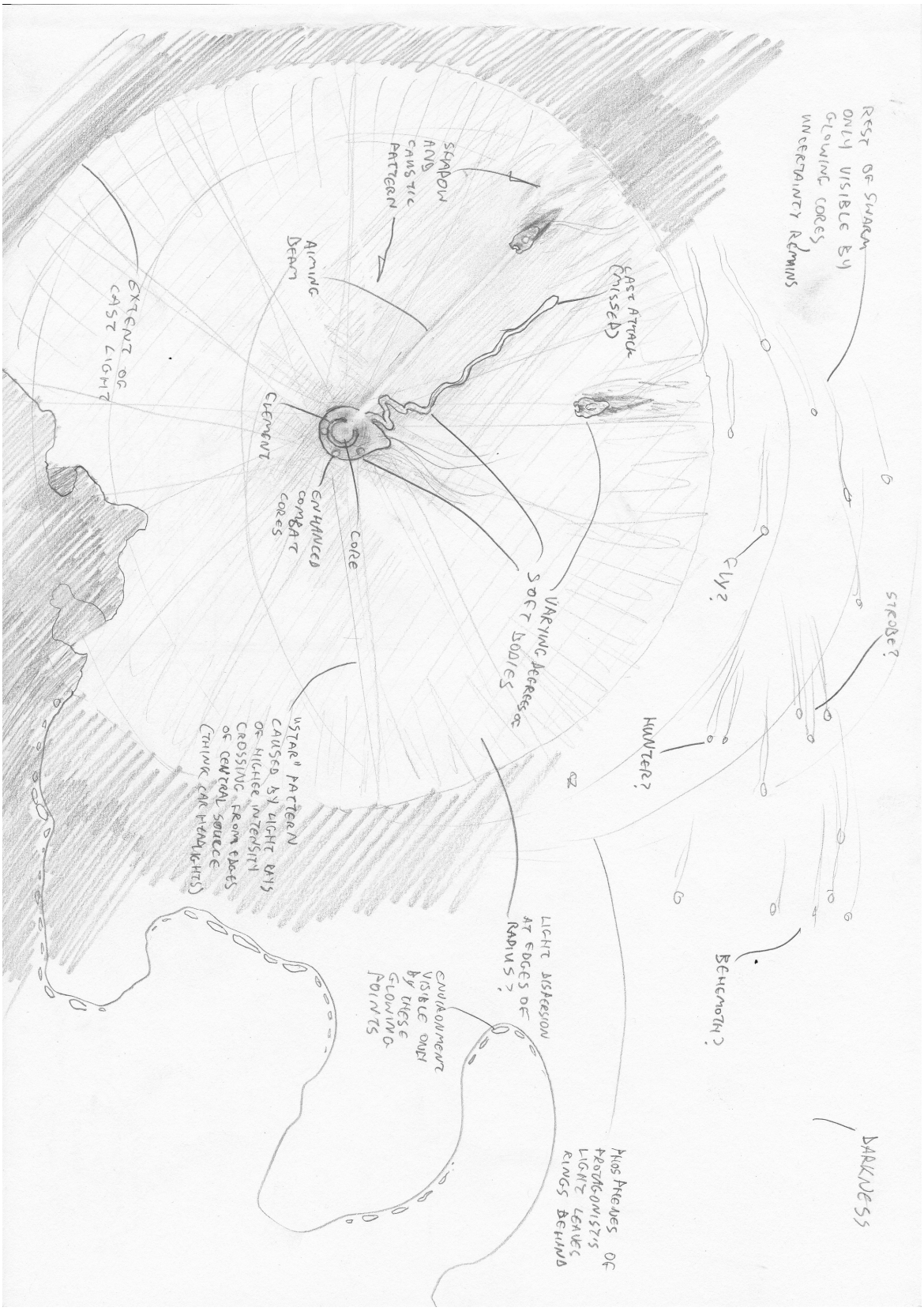
Eoghan O'Keeffe – 08543453
BSc (Hons) Entertainment Systems, 4th Year

TABLE OF CONTENTS

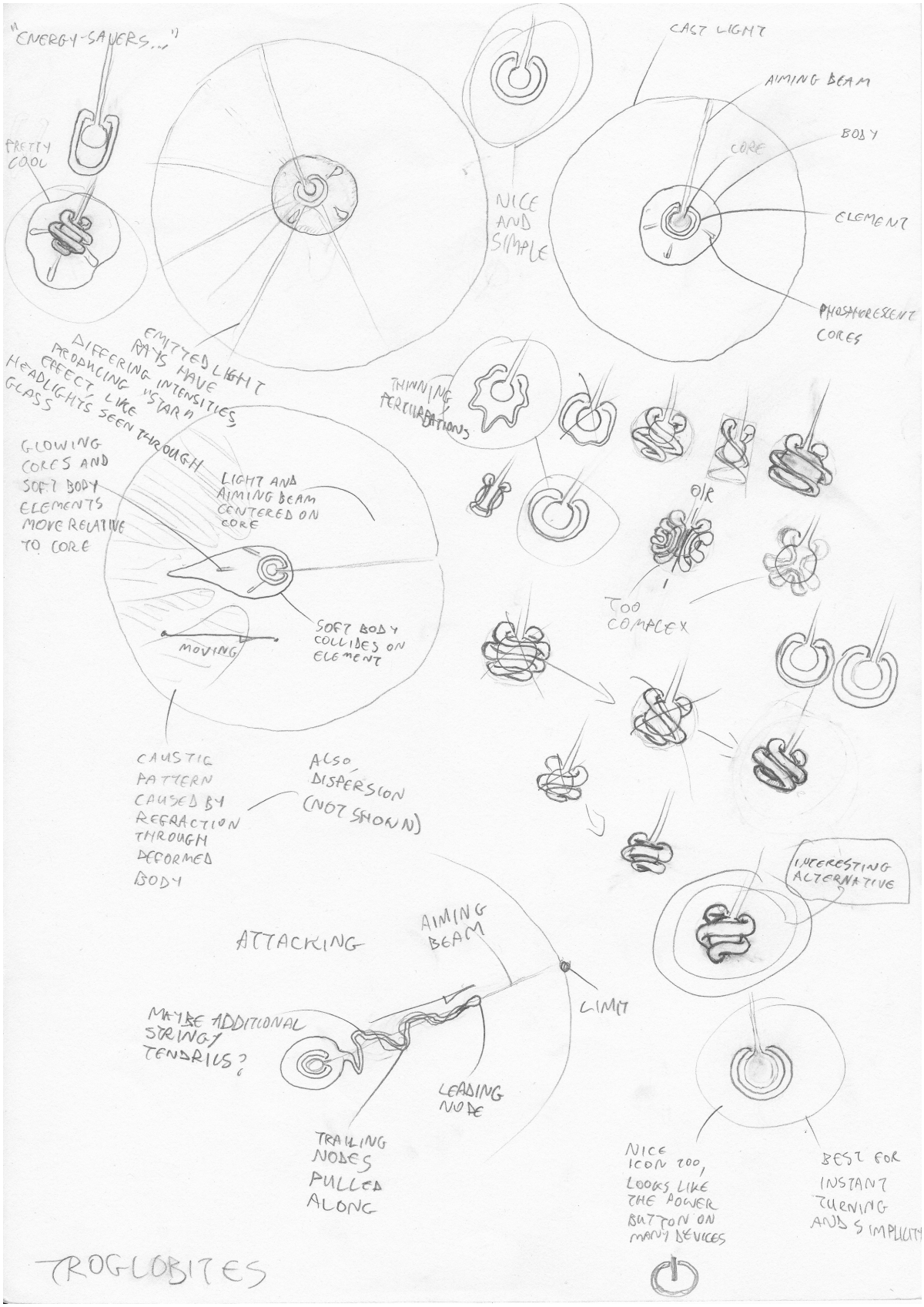
FOREWORD

This document is an addition to the preceeding report, covering areas which have arisen since its writing, and areas which have more relevance at the implementation stage of development. The design document is quite exhaustive in scope and detail, and those areas will not be covered again here, so both should be read in conjunction.

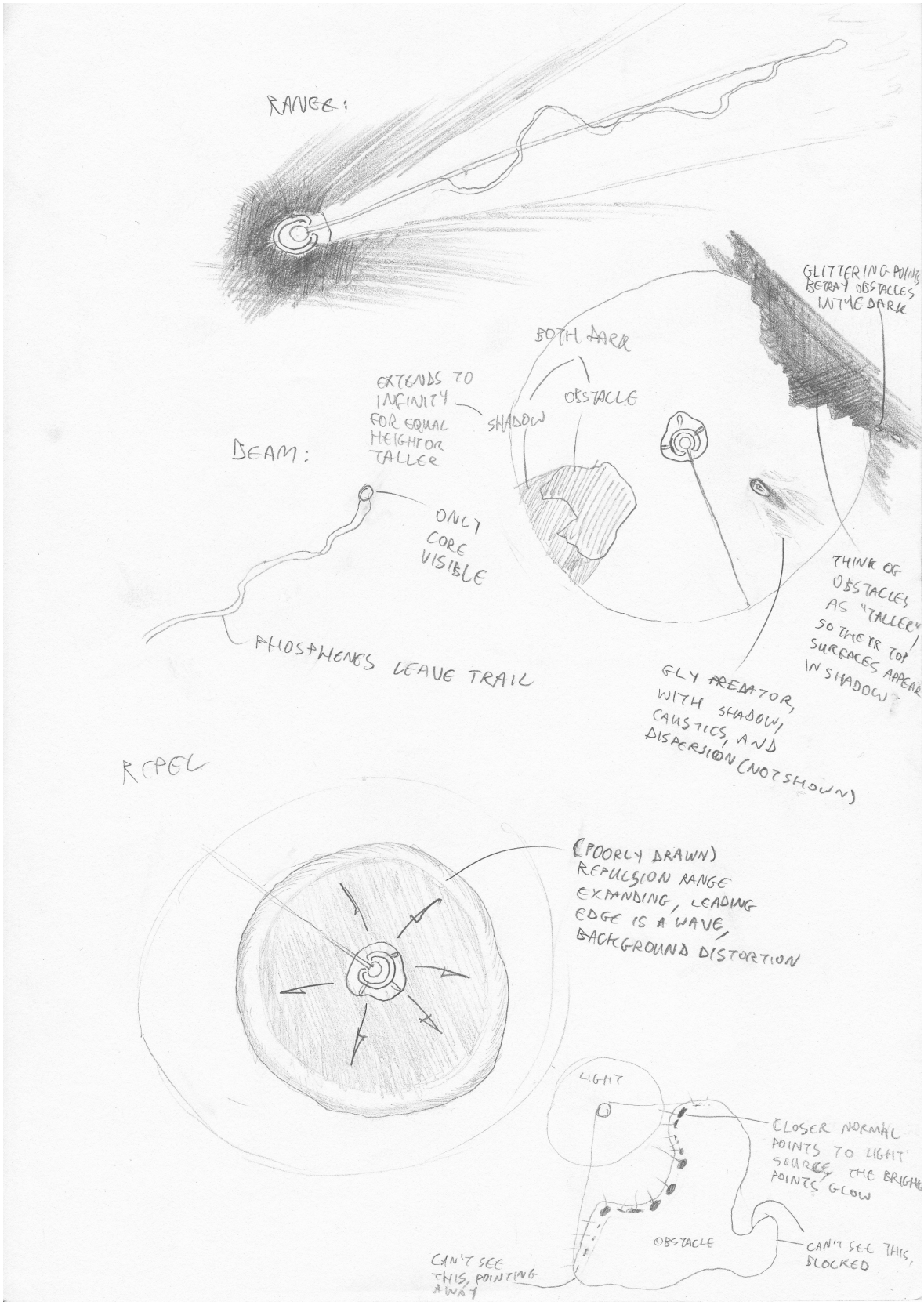
SECTION I – CONCEPT ART



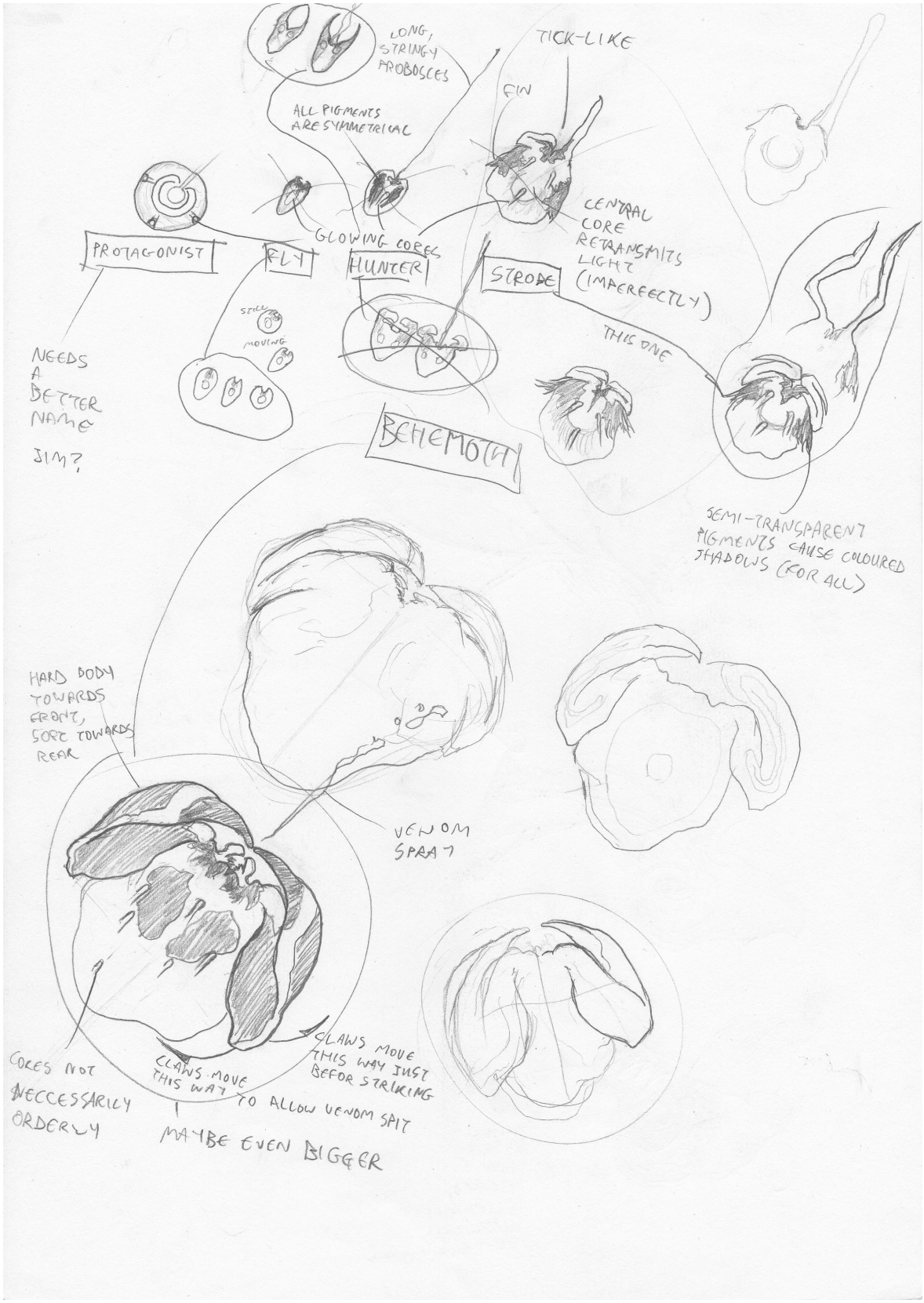
An overall view of a typical game screen



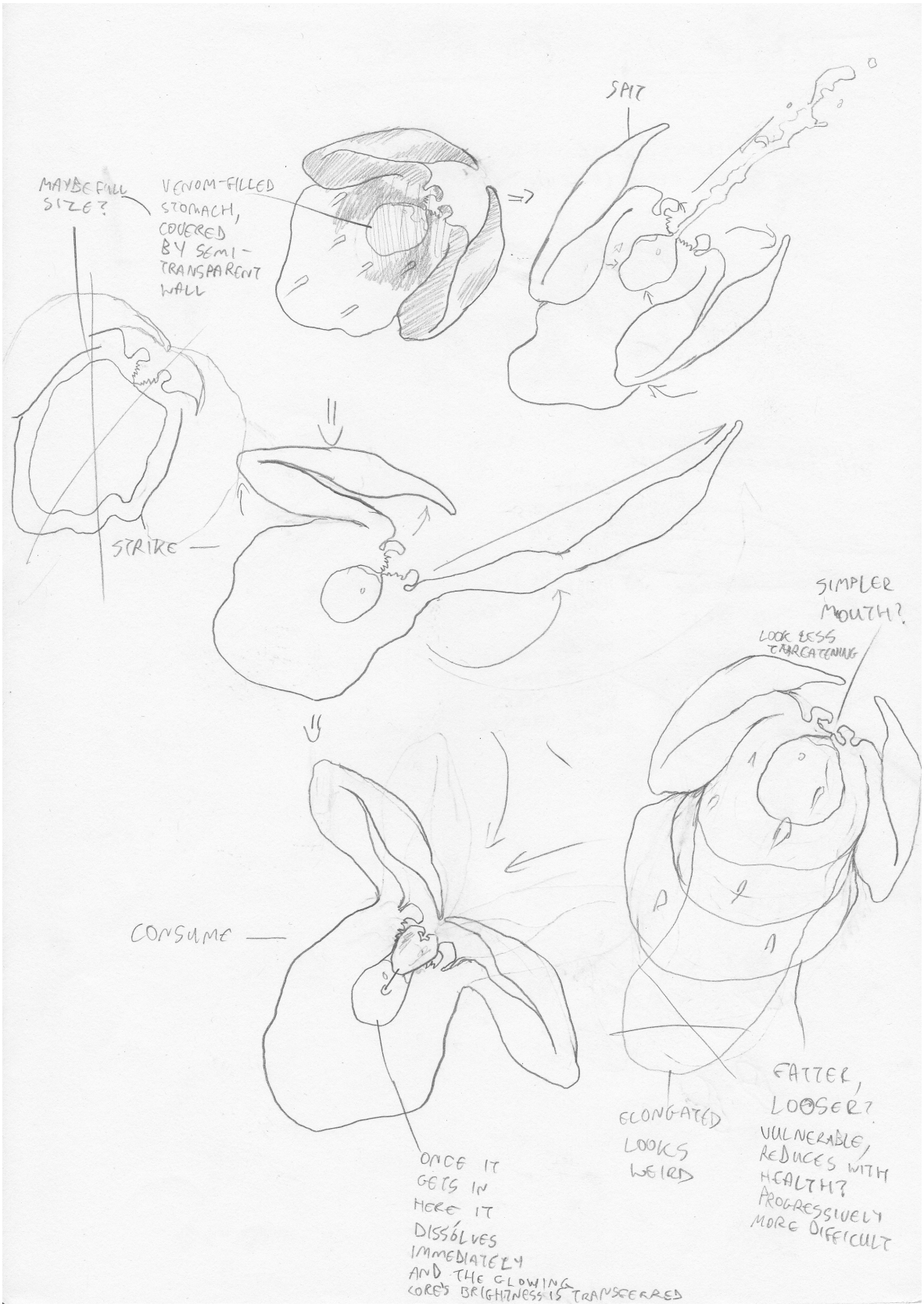
The protagonist's details



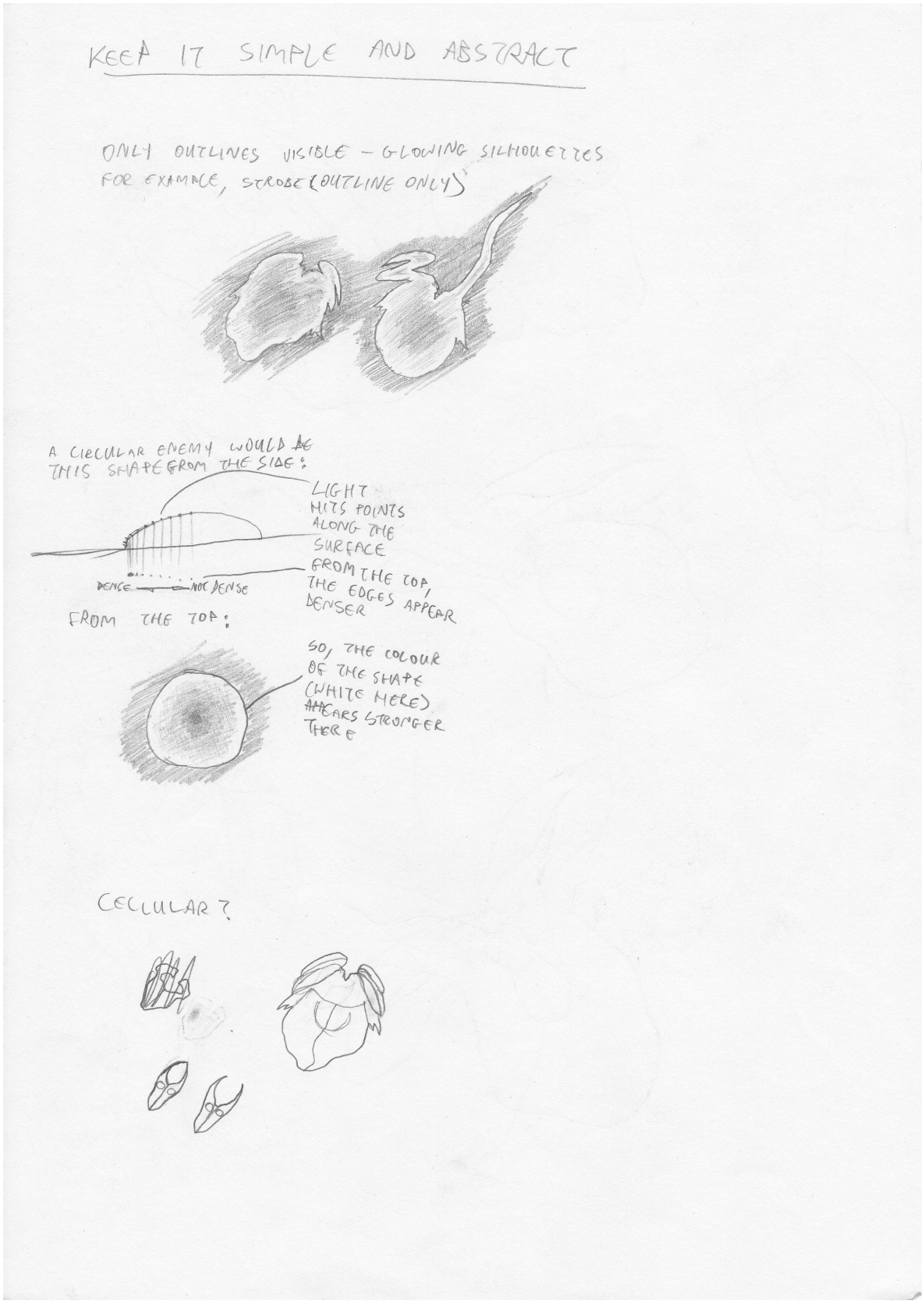
Enhanced combat and other details



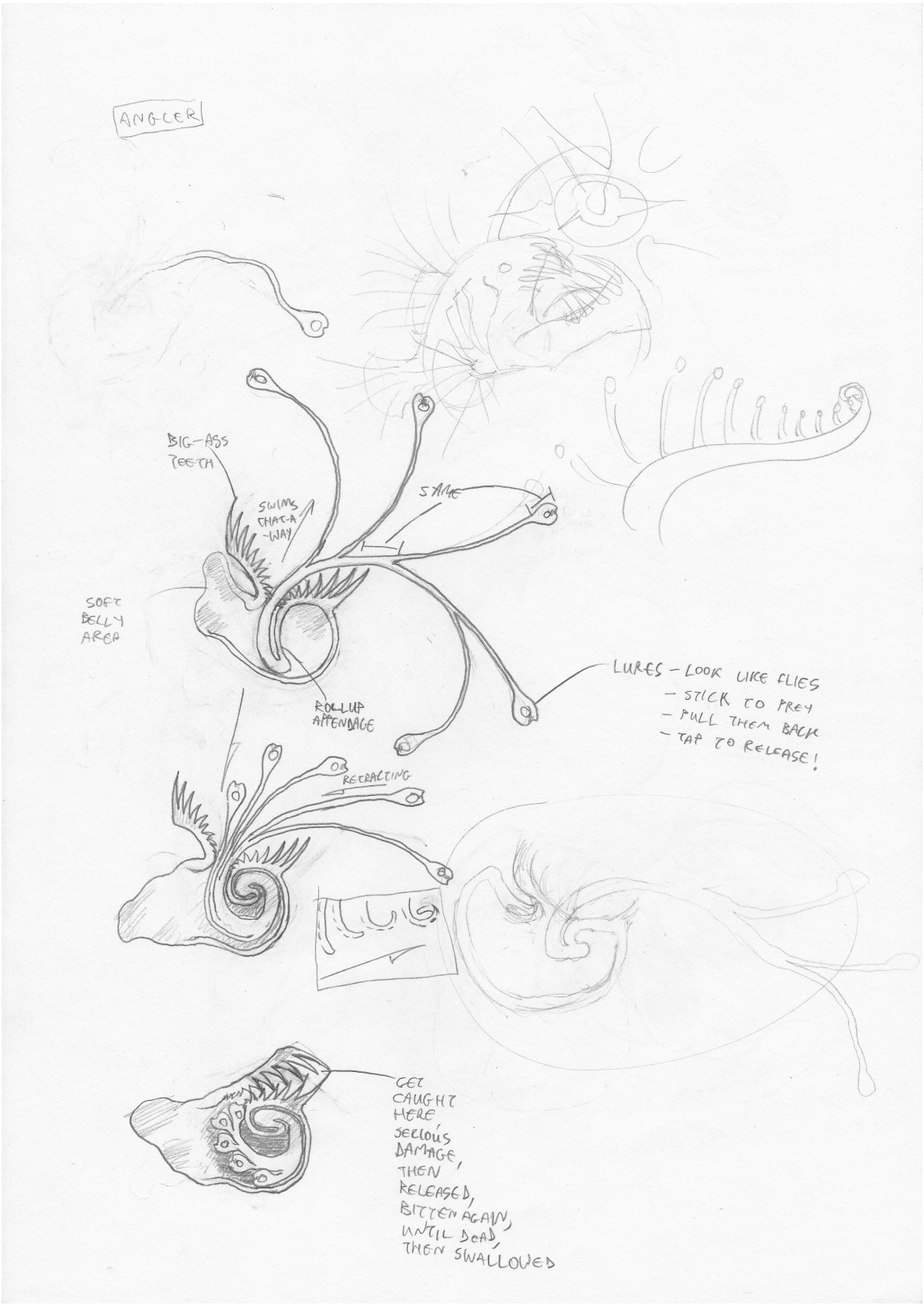
Predators and relative scales



Behemoth variations and attacks

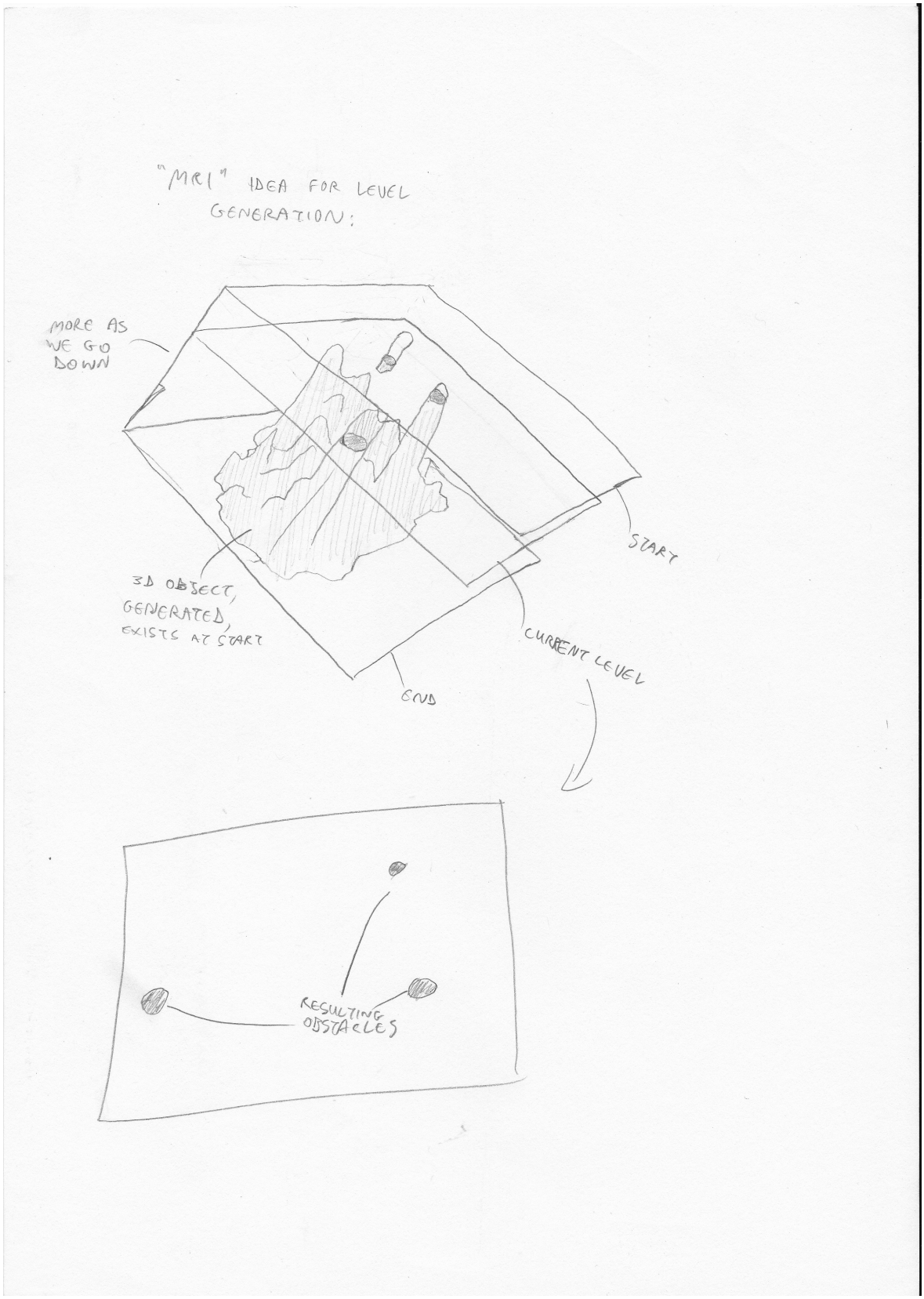


General entity look and feel



An additional predator type, for further expansion – the Angler

Fractal level generation idea



Cross-section level progression – another level-generation idea uses Conway's game of life (Wikipedia, 2011), although, due to its great sensitivity to initial conditions, it may not be possible to reliably produce similar and progressively more difficult stages

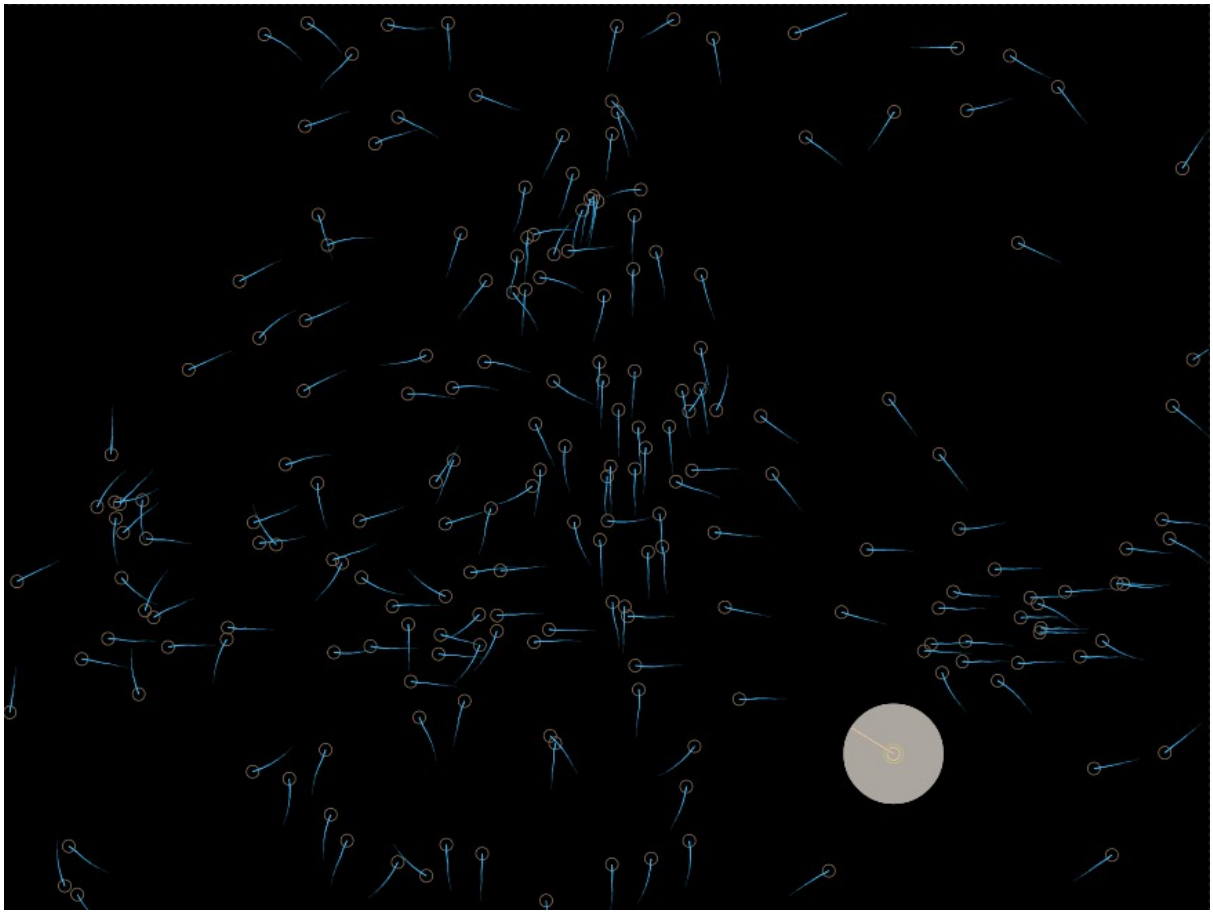
Touch screen controls

Dynamic twin sticks touchscreen controls

SECTION II – NOTES AND AMENDMENTS

1 - Review of First Prototype

Over the course of writing these reports, I developed a basic Web-based prototype to test some of the basic features – particularly the swarming behaviours. This prototype revealed some interesting issues, which informed these reports towards a more suitable design.



A screenshot of the prototype, showing predators swarming and the protagonist at the bottom-right

Firstly, the physics system used for moving the entities was originally developed in polar coordinates (derived from vectors), as it is more intuitive to direct an entity based on simple polar instructions – the magnitude of the force to be applied, and the direction (in radians) to apply it in – than with vectors.

This proved to be very problematic and counter-intuitive when it came to the force-summing methods of the flocking algorithm, and would cause further problems down the line when the rest of the physics and collision engines were to be developed.

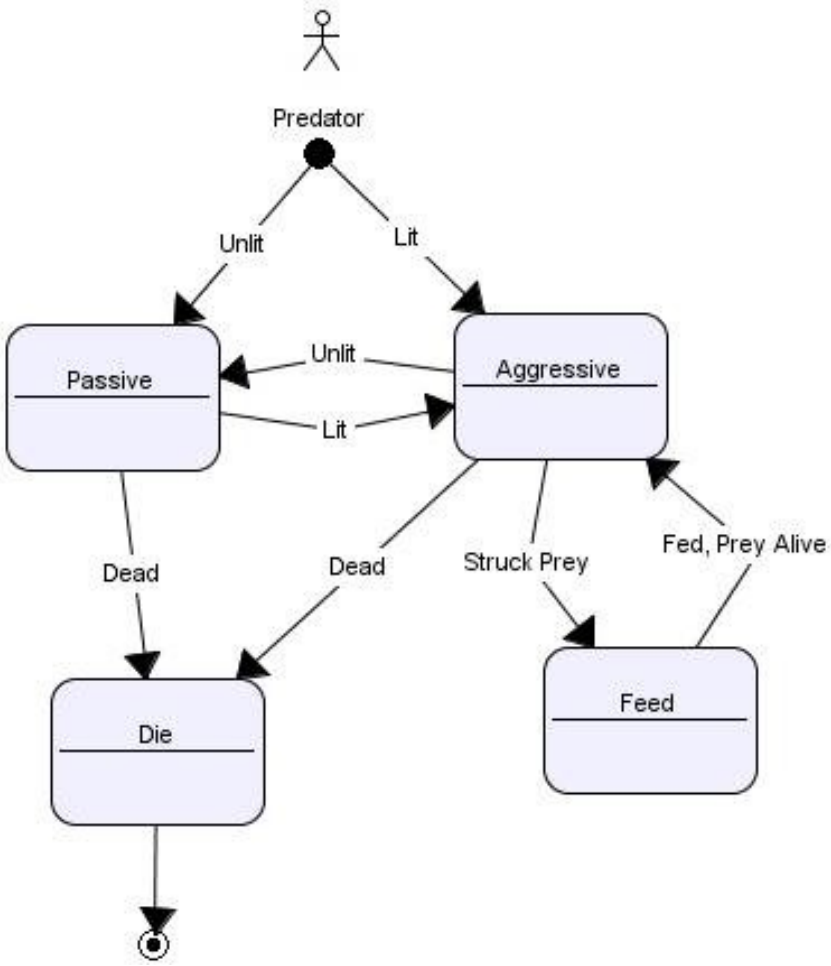
With the benefit of this realisation, and a maths module which outlined the optimal approach for vector-based physics, the next versions will use the new system.

Flocking had several performance issues, being extremely sensitive to increasing numbers of entities, and also to the size of the time step between frames. The performance optimisations part of this section (part 3) outlines some methods to address this.

Finally, due to these issues, further development of the prototype would be hindered, so I wrote it off as a lesson learned, and will move along a new path with the next versions.

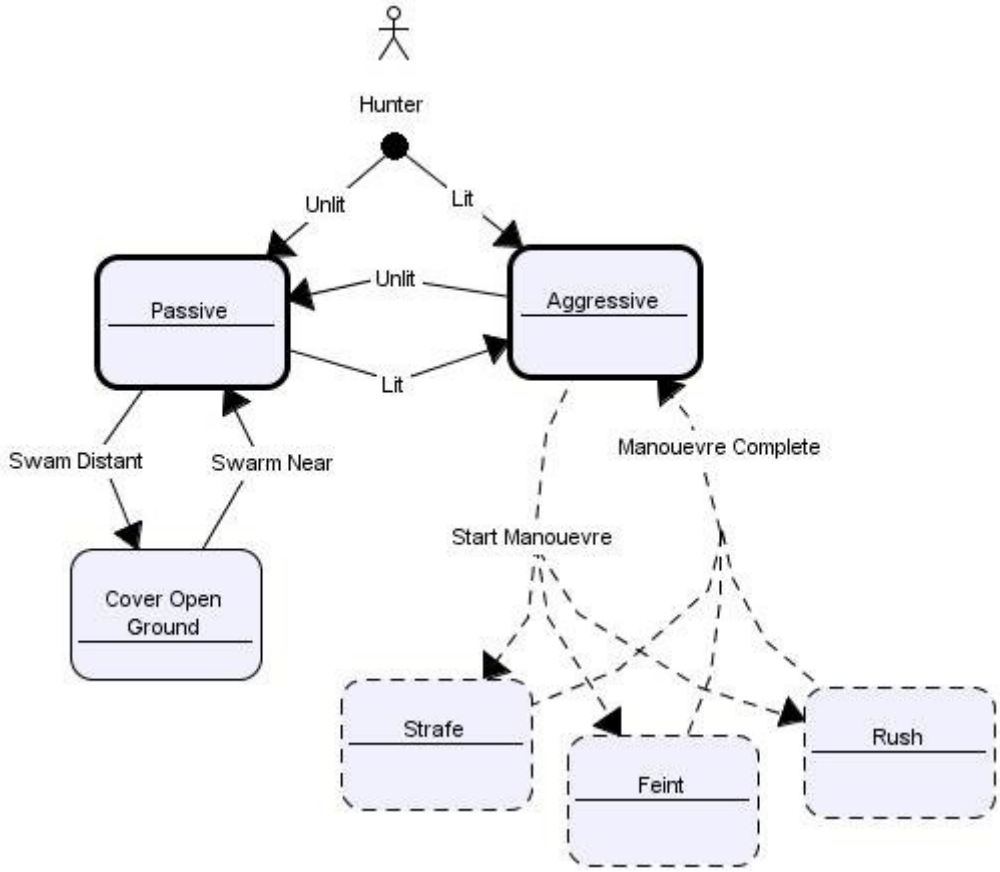
2 - State Machines for Artificial Intelligence

The following state machine diagrams better illustrate and define the processes involved in the predator behaviours. The first diagram details the processes for the global states, which every predator observes.

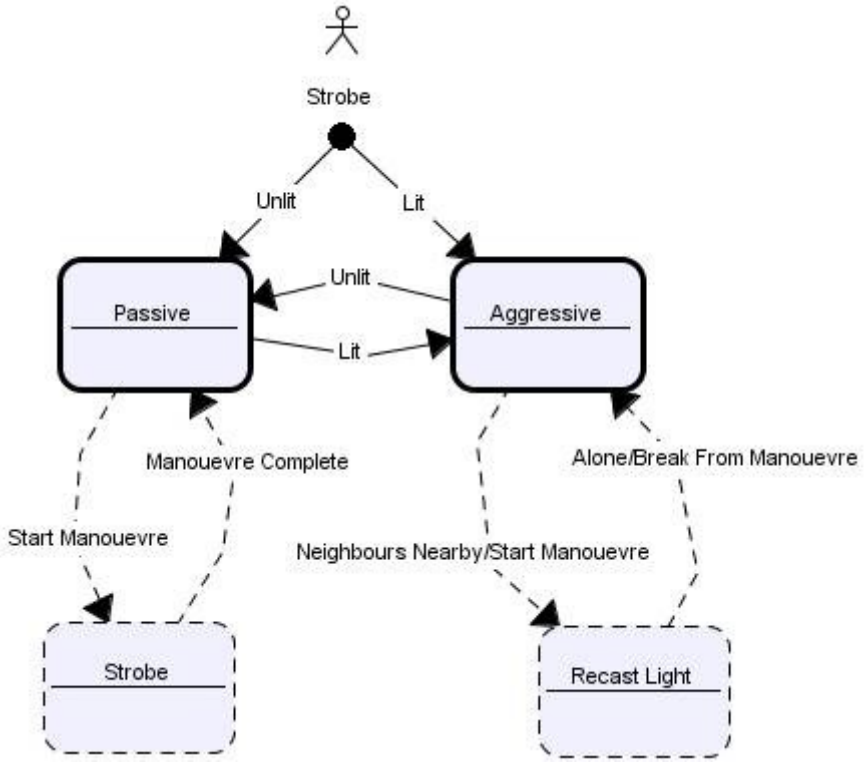


The Fly's behaviour simply observes these states, although the predators have further substates within them.

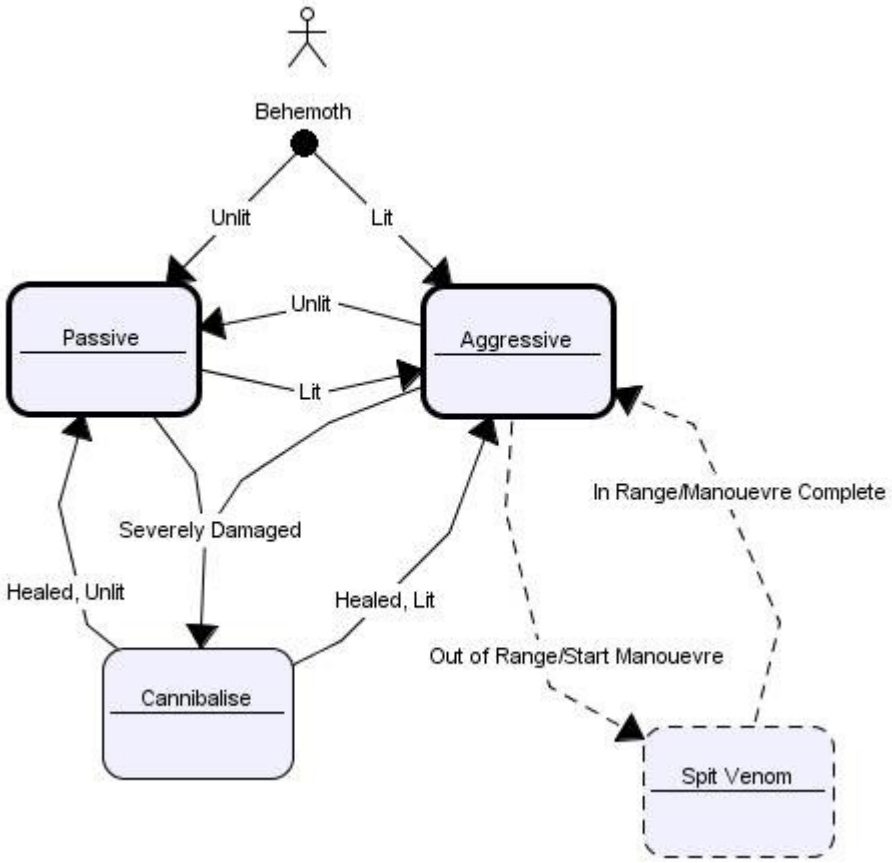
In the following diagrams, bold states represent the aforementioned global states, while normal and dashed ones represent other states and substates, respectively, within them. The rest of the global state diagram is omitted for brevity.



The Hunter's “Cover Open Ground” state both covers the maximum area of open space and ensures it returns towards the swarm – it interrupts the passive global state, but may also be implemented as a substate of the “Passive” state.
It's three attacking manouevres are minor substates, occasionally altering how the aggressive state is performed.



The Strobe's “Recast Light” state is executed for a long time when neighbours are available.



The Behemoth's “Cannibalise” state must also have a failure condition, whereby if another predator cannot be caught, it gives up and regresses back to another state.

3 - Performance Optimisations

Due to the computationally expensive nature of several components of this project – flocking behaviours, light simulation, physics and collision-detection for soft-bodied, changeable, and complex objects, as well as dynamic audio – performance optimisations must be considered. At this stage, there is no need to go into them in too much detail, just to consider them and their role in defining the architecture, strategies, and software and hardware used in developing the project.
The following are some brief notes on strategies to improve performance in these likely affected areas.

3.1 - Space-Partitioning

- A space-partitioned data structure dramatically affects several areas, particularly the flocking behaviours and collision-detection, but also any procedure which operates on objects based on their position, such as detecting intersections between light rays and objects

3.2 - Light Simulation Limitations

- Because the circumstances relating to the light simulation are known and controllable, limitations can be applied to it to reduce the number of computations required
- There is normally only one source of light in the environment (the protagonist's core), and the radius of light to be cast is also known, so querying of the space-partitioned data-structure and bounds-checking can be used to limit calculations to only the area of the specified radius around the source's position
- Light rays can be targeted at entities in the area, with the backdrop between them filled in as standard, and the more complex ray intersection operations (in particular any forward-traced ones) only taking place upon them – thus, large areas can be lit in large, simple blocks, saving time
- A combination of backwards and forwards tracing techniques can be used, with as many operations as possible being backwards-traced for efficiency, eliminating wasted casts
    - For the primary and shadow rays, at least, backwards tracing techniques can be used
    - Other operations whose effects are more complex, such as refraction and caustics, may require a forward-traced approach

3.3 - Use of Hardware

- As much applicable work as possible – such as the light simulation – is performed on the graphics card (through use of WebGL, for the web application) to boost performance and reduce the load on the processor

3.4 - Two-Phase Collision Detection

- Collision detection uses a broad phase to discover a limited set of collision tests that need to be performed based on a (possibly recursive heirarchical) series of simpler tests of bounding areas
- The narrow phase is then performed, performing the complex collision tests, reduced by the first phase (LaValle, 2006)
- This efficiency is further improved by use of the space-partitioned data structure

3.5 - Load-Balanced Operations

- In order to reduce the number of calculations done in any one frame, expensive and non-time-critical artificial intelligence operations (flocking behaviours) are carried out once every number of frames, whereas the rendering, physics, user input, and other immediate or time-critical operations are carried out every frame

3.6 - Audio Pipeline

- Serving the dual purposes of improving efficiency and synchronising playback, an audio pipeline does not play sounds immediately, but maintains a queue of sounds to be played at the next suitable time – this may bee upon an appropriate beat, or simply after all sounds in the queue have been processed and are ready to be output
- This minimises input/output calls, and provides one central instance to deal with all audio concerns

4 - Programming Style

4.1 - Abstraction

- In order to facilitate the modular architecture of the system, standard interfaces are defined for all components
- Components may change without affecting the rest of the system, as long as they implement the interface in a suitable manner

4.2 - Function-chaining

- In order to reduce the amount of code that has to be written, mutator functions (“setters”, etc.) will return the object upon which they have operated
- This allows subsequent calls to be made on that object, without starting a new line, retrieving it again, or doing other extraneous work

4.3 - Lenience

- Because this is a one-person project, there is no great requirement for strict programming paradigms to be written in order to express programmer intent

- Strictness need not impede rapid progress
- A loosely-typed scripting language like JavaScript is perfectly sufficient

5 - Software Frameworks and Libraries

5.1 - jQuery and Prototype

- One or both of these libraries may be useful in developing the web application component of this project
- jQuery simplifies many aspects of web development, and is the de facto JavaScript library in the industry (jQuery, 2011)
- Only a subset of the overall library is of principal interest for this project
    - Event handling API
    - Basic utility methods
    - Rapid DOM traversal and manipulation
    - AJAX and XHR methods
- The Prototype library (Prototype, 2011) offers a similar set of functionality to that of interest of jQuery, with additional support for OO, class-driven development, extending the capabilities of JavaScript – this is of great importance if the lack of "true" OO paradigms in current JavaScript implementations is a significant obstacle
- The alternative to using either of the above two libraries is to use raw JavaScript – this is more difficult and takes longer, but reduces the project's dependancies and makes the code more portable

5.2 - Modernizr.js

- Modernizr is a feature-detection library (Modernizr, 2011) for web applications
- Although the game itself may largely be a "take it or leave it" application (in that if a feature isn't supported, it can't be run), some fallbacks may be provided for certain components in the modular structure, and feature-detection is nonetheless needed to inform the user about the browser feature requirements

5.3 - HTML5 Boilerplate

- Due to the wild disparity in supported features across internet browsers and thier many various interpretations of the W3C standards, a code base for web applications is often required to standardise these differences and provide a reliable wrapper for the rest of the application – HTML5 Boilerplate provides this cross-browser consistency (HTML5 Boilerplate, 2011)
- It standardises many areas in HTML and CSS, and provides the aforementioned jQuery and Modernizr (as well as other tools, in custom builds) to provide a generic, best-practice normalised base for the application

5.4 - Fabric Engine

- This engine provides the speed of multi-threading and compiling for the single-threaded, interpreted JavaScript (both client- and server-side) (Fabric Engine, 2011)

- Considering the heavy load of the processes involved in this project, these performance enhancements could be vital

5.5 - Three.js

- This is a JavaScript 3D library, greatly simplifying the work of 3D rendering (Cabello, 2011)
- Its suitability depends largely on how invasive its simplifications are to the required features of the project

5.6 - Node.js

- A server-side JavaScript library, Node is a recent alternative to using PHP, Java, and so on (Node, 2011)
- Its use here is mainly concerned with implementing a sever for user profiles and scores

5.7 - Sony PSP Development Libraries

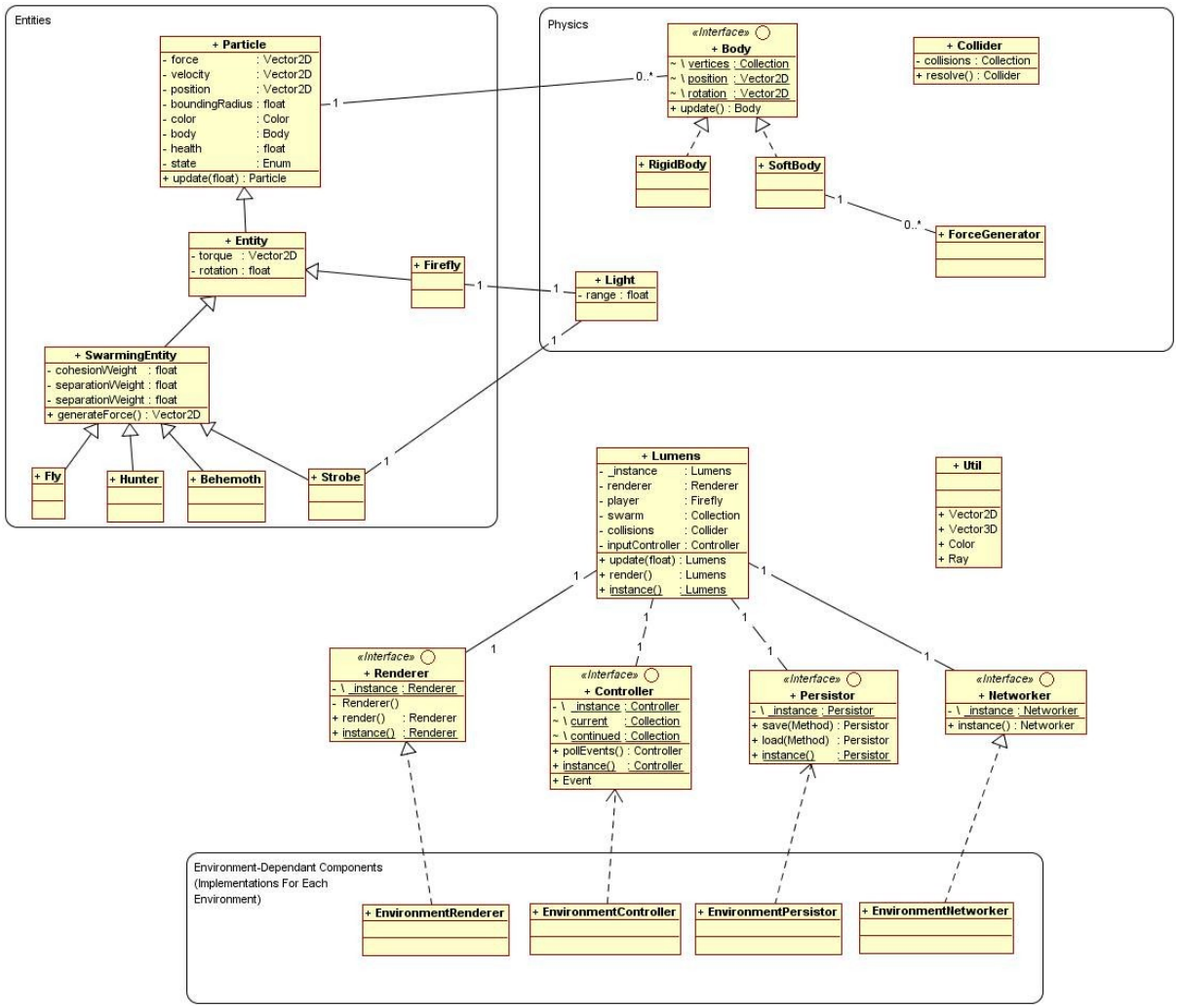
- For the PSP development stage of the project, the official Sony PSP development kit will be used, along with the accompanying plugins for the Visual Studio 2008 environment

5.8 - Cocos2D and Box2D

- These two frameworks would greatly simplify the development of trhis project for supported platforms – though they may interfere with the development of certain features
- Cocos 2D is a game development framework, supporting C++, JavaScript, Objective-C, and Python (Cocos 2D, 2011) – there is also a 3D version available
- Box 2D is a 2D physics engine, integrated in Cocos 2D (Box 2D, 2011)

SECTION III – SYSTEM ARCHITECTURE

The UML diagram below (developed using Open ModelSphere) describes a suitable system architecture:



1 - Cross-Platform Considerations and Modularity

In order to accomodate the iterative nature of development and the platform-dependant aspects of the project cleanly, the components of the system architecture should be as modular and loosely-coupled as possible.

2 - Environment-Independant Components

2.1 - Physics

- Manages the phsyical simulation of the environemnt

2.2 - Artificial Intelligence

- Controls the predators and protagonist's wandering behaviour

2.3 - Game Logic

- The overall management of the game, its occupants, main loop, states, menus, etc.

3 - Environment-Dependant Components

3.1 - Rendering

- Because the light simulation renders a fully populated scene, objects are retrieved and rendered by this component itself – they do not render themselves

3.2 - User Interface

- This component standardises user input from any of the various devices into an easily-queryable data structure
- For touch devices, it requires access to the rendering component in order to draw its own on-screen indicators

3.3 - Persistence

- Manages saving and loading on the platform in question – for the web application, this means using the best form of persistence available (HTML5 Local Storage, for example, may be very helpful)

3.4 - Networking

- Provides a simple interface for communicating over the internet
- Manages the lifecycle of any network access

REFERENCES

Wikipedia, 2011 (online), en.wikipedia.org, ”*Conway's Game of Life*”, accessed 30.11.2011, URL: http://en.wikipedia.org/wiki/Conway's_Game_of_Life

LaValle, Steven M., 2006 (online), “*Planning Algorithms*”, “*5.3.1 Basic Concepts: Two-phase collision detection*”, planning.cs.uiuc.edu, accessed 28.11.2011, URL: http://planning.cs.uiuc.edu/node214.html

jQuery, 2011 (online), www.jquery.com, accessed 28.11.2011, URL: http://jquery.com/

Prototype, 2011 (online), www.prototypejs.org, accessed 28.11.2011, URL: http://www.prototypejs.org/

Modernizr, 2011 (online), www.modernizr.com, accessed 28.11.2011, URL: http://www.modernizr.com/

HTML5 Boilerplate, 2011 (online), www.html5boilerplate.com, accessed 28.11.2011, URL: http://html5boilerplate.com/

Fabric Engine, 2011 (online), www.fabric-engine.com, accessed 1.12.2011, URL: http://fabric-engine.com/

Cabello, Ricardo, 2011 (online), www.mrdoob.com, accessed 28.11.2011, URL: http://mrdoob.com/122/Threejs

Node, 2011 (online), www.nodejs.org, accessed 28.11.2011, URL: http://nodejs.org/

Cocos 2D, 2011 (online), www.cocos2d.org, accessed 28.11.2011, URL: http://cocos2d.org/index.html, http://www.cocos2d-iphone.org/, http://cocos2d-javascript.org/about

Box 2D, 2011 (online), www.box2d.org, accessed 28.11.2011, URL: http://box2d.org/about/