**SD3 Coursework Planning**

Jordan Stephano Gray (40087220)

Brief

* Use design patterns, threads and a GUI create a game.
* Game is tile based.
* Game has move, GUI and able to undo moves.
* Consists of a Level (Tiles), Playable Character and enemy NPC’s.
* Default grid is 4x4. Can change grid size.
* At the start, player is placed on any tile except top left.
* There are 3 main food sources, one of them being the enemies. The players characters diet changes dynamically.
* Can move to any adjacent square (Even diagonally).
* Cannot move outside tiles.
* The player character moves at random when a button is pressed.
* 3 main enemy NPCs.
* Can generate enemies at random.
* No one leaves the tiles.
* Enemies move the same as the playable character.
* Every time the player moves there is a 1/3 chance of a new enemy spawning. Type of enemy is random.
* Can create more enemies.
* If player moves into the same tile as enemy the enemy is killed and removed.
* If player moves into tile with 2 or more enemies, the player will die unless his diet is set to “enemies”.
* Game over when player dies.
* If players diet is “enemies” and moves into same tile as 3 or more enemies the player still dies.
* Game controlled from GUI.
* Create a report explaining where in the code each design pattern and threads are used.
* Highlight potential problems if not using design patterns.
* Report will discuss functions of the GUI and use creenshots.
* Weighting:
* Creativeness(10%)
* Patterns(Factory(8%), Observer(8%), Strategy(8%), Command(16%), Total(40%))
* Threads(10%)
* GUI(20%)
* Variable Tile Size(10%)
* Report(10%)

Initial Ideas

* A sc-fi setting based on a ship.
* Player character plays as a man-eating alien.
* Outside tile is the entry point into the ship.
* Diet: Small = Space Rodents, Medium = Space Dogs, Large = Enemies
* Increase in tiles means a larger ship.
* Enemies: Humans, Deathclaws, Giant Space Bees

Initial Classes

* Player Class
* Enemy Class
* Human Class Extends Enemy
* Deathclaw Class Extends Enemy
* GiantSpaceBee Class Extends Enemy
* GUI Class
* Level Class
* Tile Class

Where Patterns Fit in

**Factory Pattern:**

Takes in user input and selects from n possible variables (i.e Colour) and whatever is chosen is what the object is instantiated with.

**Use:**

Can use to spawn 1 of 3 enemies OR

Use to spawn the player with 1 of 3 possible attributes.

**Observer Pattern:**

Loosely coupled designs between objects that interacts. Observers Observe Observable Objects. Example, Student is an observer and Lecturer is observable, Students must know how to update themselves when new info is supplied by the lecturer.

**Use:**

**Strategy Pattern:**

Example, 2 types of ducks that can swim and quack and fly. When introducing rubber duck and decoy duck which cannot carry out some of these functions rather than overriding methods you can use interfaces. All these ducks can swim so that stays in the duck class but quack and fly are implemented using interfaces. Under these new interfaces you can create a variety of behaviours e.g for QuackBehaviour interface have 3 classes use it called “Quack”, “MuteQuack” and “Squeak”. Now the main Duck class can use this new interface(s) which allows it to set any ducks behaviour which inherit from the Duck class e.g “setQuackBehaviour(in quackBehaviour, QuackBehaviour)”.

**Use:**

**Command Pattern:**

An object is used to represent and encapsulate all info needed later on. Useful for organising and managing tasks, wraps 1 or more actions into a single command object. Uses include:

* Wizards
* Thread Pools
* Parallel processing
* Anywhere there is a queue of tasks needing execution

E.g code a restaurant. A queue of tasks being food orders, waiting to be executed cooked. Create a food order to be passed to a waiter (queued) to then be passed to the chef later.

**Use:**

Could pass along each entity on the screens movement to a movement queue to then be executed maybe even via threads. This requires a movement handler.