OO A1

Notes and documentation

Team Name: The Zappers

Key Info:

Task	Date	Weight	Place
Project milestone I: initial design	**Friday 28th April**	15%	GitHub
Project milestone II: implementation	Monday 8th May	15%	GitHub
Project milestone III: additional features	Monday 22nd May	20%	GitHub

Javadoc problems and their solution:

- Javadoc generation creates a cascade of errors and warnings. To fix this, when you select "Generate Javadoc" in Eclipse, rather than clicking "Finish" click the "Next" button twice to go to the third page of the dialog. In the "Extra Javadoc options" box, enter the following text:

-Xdoclint:none
Then click "Finish". The Javadoc should then successfully generate.

Contents: (click and follow link)

Stage 1

- Show only the new parts
- Show where new classes fit into existing system
- **Must** contain names of methods within class - just the new ones
- Names and signatures not big deal, likely to be refactored
- Overall responsibilities of the class must be documented
- Must **write a design rationale** explain the choices you made. You must explain both how your proposed system will work and why you chose to do it that way.
- Create sequence diagrams, for instance for:
    - Force users mind control
- Design must include
    - Classes that exist
    - Role of new or modified classes
    - How classes relate
    - How classes are used to deliver functions
- UML Diagramming Tools / By Pen & Paper?
- Save multiple versions of design doc
- Submit a WBA (Work Breakdown Agreement)
    - Who's responsible for producing each deliverable
    - Who's responsible for testing or reviewing each deliverable
    - Dates deliverable, review and test need to be submitted

UML notes
- Ctrl + Alt + mouse wheel to zoom
- Use the below as a template for all boxes
- more shapes bottom left, you can select UML templates (delect all others for ease)
- bottom of UML shapes is all the arrows we need

WHEN DONE, ALSO COMMIT THE EXPORTED THING TO THE FOLDER ON GITHUB
ALSO ADD IN HANDDRAWN SKETCHES TO THE FOLDER TOO

A1 Project requirements

EVERY ONE OF THESE THINGS SHOULD BE COMPLETED AS A COMMIT.
NEVER SHOULD MORE THAN ONE THING BE COMPLETED IN A SINGLE COMMIT.

- **Add The Force**
  - Designate people the ability to use the force
  - People with a little bit of force ability can resist Jedi mind control
  - People with a lot of force ability can control minds (of non force people) - move them in a direction of mind-controllers choice
- **Lightsabers**
  - Anyone can pick a LS up
  - Only people with the force can wield it
  - Therefore, only people with the force can use it as a weapon
- **Ben Kenobi's Training**
  - Ben can train luke
  - Training luke raises his force ability
  - When trained enough, Luke can use a lightsabre
- **Droids**
  - Droids cannot use the Force
  - Droids have specified owners
  - Droids follow their owners if there are no obstructions (wall, etc)
  - If Droids try and move to the Badlands, they lose health
  - Droids regain health if they use oil
  - Droids regain health if another person uses oil on them.
  - Droids' become immobile on no health
  - Immobile Droids can be disassembled into Droid parts.
  - Some people know how to repair Droids that are immobile.
  - Droid parts can be used to repair immobile Droids
  - Droid parts that are used on immobile Droids cannot be used again
- **Healing**
  - Can drink from a canteen
  - Drinking heals the drinker a little bit
  - Droids regain health when oil can is used on them - by themselves or others
  - Oil can has multiple uses - one use doesn't deplete oil can
  - Canteen can be refilled
  - Oil Can cannot be refilled
- **Infrastructure**
  - Do not modify the code of the engine! **Ever**! Never do it!
  - Specifically, packages starting in **edu.monash.fit2099**

Submission

Will be marked on the **MASTER BRANCH**
Put design documents in the *design-docs* folder in our private repo.
Use the repo given to us by tutors.

Marking Criterion

Design completeness
Does your design support the functionality we have specified?

Design quality
Does your design take into account the quality properties and heuristics we have discussed in lectures, including:

- Not Repeating Yourself.
- Encapsulation.
- Information hiding.
- Cohesion (modules within the system have a clear, easily described purpose).
- Dependency control - have you minimised dependencies between modules, and made them explicit and easy to understand?

Practicality
Can your design be implemented as it is described in your submission?

Following the brief
Does your design comply with the constraints we have placed upon it - for instance, does your design leave the engine untouched, as required?

Documentation quality
Does your design documentation clearly communicate your proposed changes to the system? This can include:

- UML notation consistency and appropriateness.
- consistency between artifacts.
- clarity of writing.
- level of detail (this should be sufficient but not overwhelming)