Project title (include tutorial number?)

Your name

# Specification

*-2 sentence summary, -screen mockups (intro, game screen, game over screen), -english plain text detailed description of what happens in the game from start to finish, -describe any special events (boss characters etc), -list assets (sprites music sound fx), -structure of the game (how to win, levels, progression, difficulty)*

# Technical design

*This is a technical language and technical diagram heavy explanation of HOW you are going to implement the specification. For a simple tutorial assignment, you want one or two appropriate diagrams or algorithms (flow chart, pseudocode, UML diagrams(level5), etc.). There are lots of little tutorials, so we don’t want something excessive for each. Over all your apps/projects we should be able to see a couple of examples of all the technical design approaches/diagrams we’ve covered. If this was a big game project however, then this would span multiple pages with descriptions of code, diagrams, pseudocode, etc.*

# Test plan

*How do you know when to stop, how do you know it works, is it working appropriately (fun), etc. Again, for a small tutorial assignment, this would probably the classic basic table of test input and what you expect it to output. For larger programs this could include instructions on different tasks the user is meant to perform, and how they are meant to work – imagine you were writing instructions for a test department with staff who don’t know you or the code. For a big game project it would include instructions on what the player is expected to feel, is this bit meant to be easy, are they supposed to be confused, is this bit meant to be hard, how long is it meant to take to play this bit, etc. Then you play test, observe and test. For big projects you’d use an issue tracker (bitbucket) to record information about all these things and figure out which are important and need fixing and which aren’t (it’s not complex, just a shared online repository of bugs with priorities). It’s part of the classic software development lifecycle, design->implement->test->repeat until you get something really good. Fail fast, fail early. Find the optimum solution. For a large project you’d include user testing, where you watch someone play and make notes about what they liked, didn’t like, where they got stuck, if something broke, balance issues – things too easy or hard, things too confusing, etc.*

# GiT commit log

*All work should be kept on GiT, bitbucket and github are free to use. Make sure the repository is marked private or people will google the code and find it. A screen shot of the git commit log will suffice, it needs to show who did what and when. At level 4 it will take a while to learn to use GiT, but we will eventually.*

# Schedule

*Apply some common sense, if it’s a simple tutorial item, if it’s small, then a basic bullet point list is fine. So, tutorial assessment, Joe wants to get it finished for the week after next session. Sets aside 4hrs: 0.25hr spec, 0.5hr design, 3hr implement, 0.25min test, 1hr slack. See how long it really takes, next time adjust accordingly.*

|  |  |  |
| --- | --- | --- |
| ***Task*** | ***Estimated Hrs*** | ***Actual Hrs*** |
| *Spec* | *3* | *4* |
| *Design* | *3* | *4* |
| *Implement* | *14* | *18* |
| *Debug and test* | *1* | *2* |
| *Slack* | *3* | *5* |
| ***total*** | ***24*** | ***33*** |

*Milestones*

1. Get map working
2. Get snakes head on map moving around
3. Tail gets longer per food
4. Food appears randomly over time
5. Score on screen
6. Intro screen with instructions
7. Game over with high scores
8. Final test