 **CP2406 Coding Project Brief**

This document has been prepared by Jason Holdsworth for James Cook University. Updated 28 April 2019.

**© Copyright 2018**

This publication is copyright. Apart from any fair dealing for the purpose of private study, research, criticism, or review as permitted under the Copyright Act, no part may be reproduced by any process or placed in computer memory without written permission.

# Purpose (Assessment Tasks 1 & 2 Overview)

You are given a programming challenge that consists of frontend GUI Java programming and backend application logic programming using appropriate Java APIs and your own Java class code. The first part of the coding project assesses your ability to interpret project requirements and start developing a software solution to the challenge. You are expected to: 1) construct appropriate user stories based on the challenge requirements, 2) design a UML class diagram for the project, 3) establish a GitHub repository for your work, and 4) start developing a coded prototype based on the UML class diagram that implements the important user stories central to the incremental development of your software solution. The second part of the coding project assesses your ability to continue implementing the coded prototype based on the project requirements.

The quality of your code is assessed in terms of: 1) general readability, 2) application of Java coding standards, and 3) your ability to apply good coding practices as discussed during the subject.

# Project Description (aka Customer Requirements)

The game “Light cycles” comes from the classic Disney movie “Tron”. In the movie, each user rides a bike called a “light cycle” over a field called “the grid” (Figure 1). The objective of the game is to be the last user still riding a light cycle on the grid.

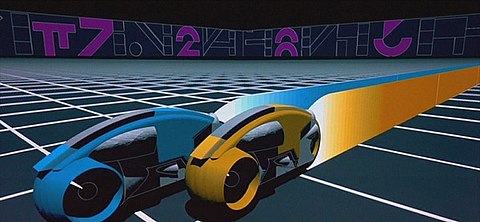
[](https://en.wikipedia.org/wiki/File:Tron_Lightcycles.jpg)

Figure 1: A close-up scene of two light cycles racing each other from the movie.

You are creating the classic arcade version of the game from the first Tron movie (Figure 2). The game is played on a 2D top-down view of the grid with light cycles and jet-walls. When a new game is created, the grid has a fixed size 1000px2.

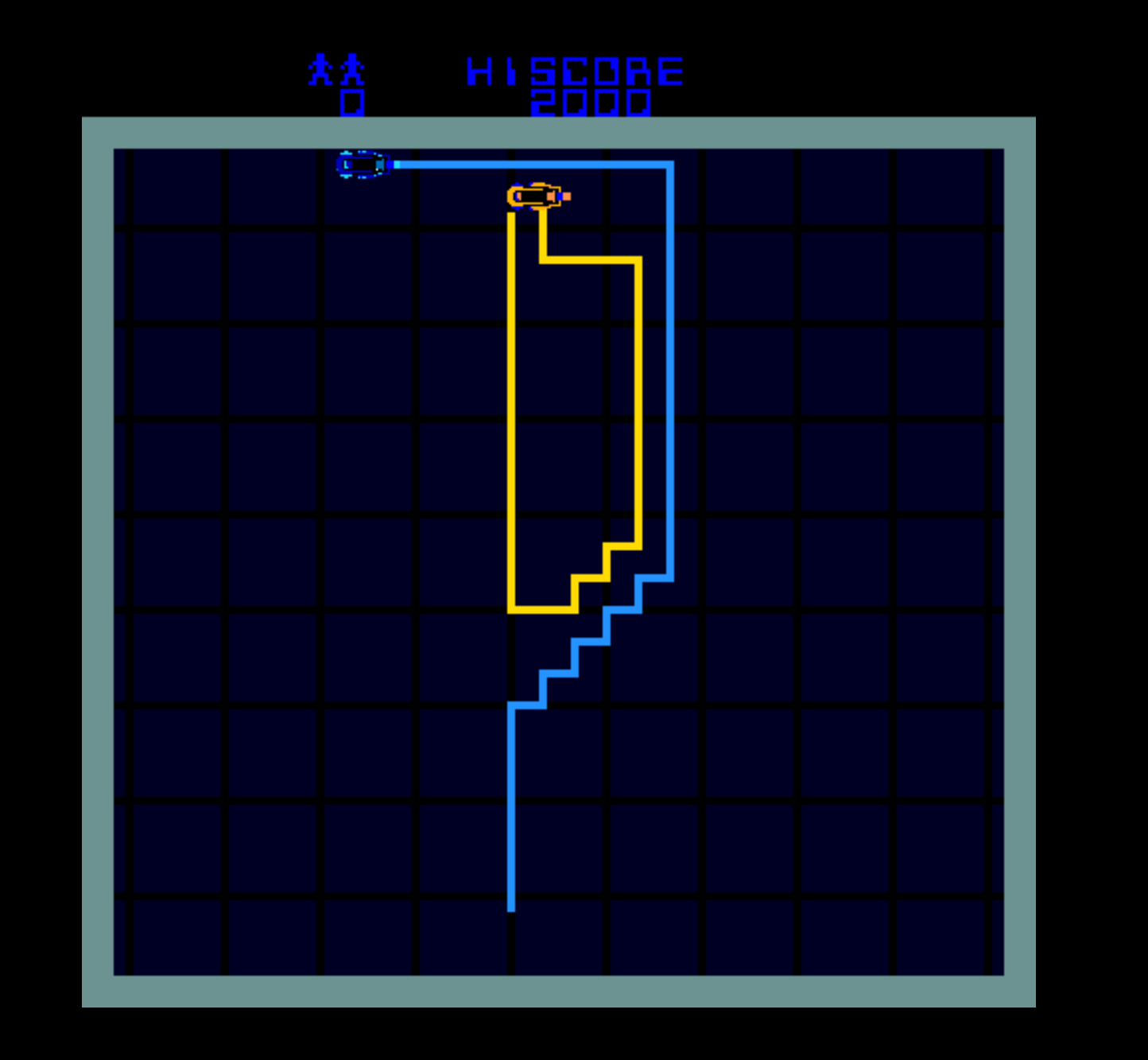
[](https://youtu.be/ONg0rUogiEg)

Figure 2: An example screenshot from the arcade version of light cycles.

This image is hyperlinked to a YouTube video of the game in action.

Each user controls their light cycle by moving it around the grid in vertical and horizontal straight-line paths. The direction of a light cycle instantaneously rotates 90 degrees when turned clockwise or anticlockwise by the user. In addition, a light cycle can speed up and slow down. However, once started a light cycle won’t stop again.

A light cycle produces a “jet-wall”, a continuous and impenetrable barrier that trails behind the light cycle as it moves around the grid. A user can switch the jet-wall of their light cycle on or off at any time during game play. When a light cycle hits a jet-wall or the edge of the grid it immediately “derezzes” and the user riding it is out of the game.

It’s not clear from the movie what the maximum number of users are that can add themselves to the grid before a new game starts (so, see below for the expected program behaviour).

The winning user gets their name and score automatically added to a “leader board” for the game. The score awarded to a winning user is calculated as the length of the jet-wall they created during game play in pixels.

# The light cycles data-protocol

You are required to implement the data-protocol as described here to handle multiple users playing a single game together. This is achieved by creating a single “server” program that handles requests from many “client” programs (see Figure 3).

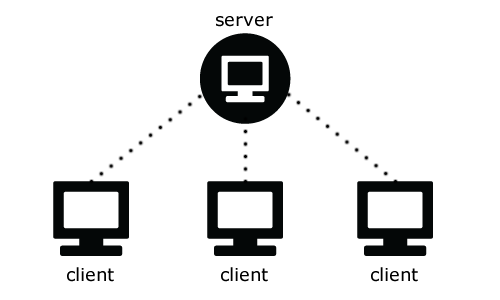


Figure 3: Example of a server handling 3 clients simultaneously.

The server program maintains the game state and user information. It broadcasts data in the following format to all clients while a game is being played: **name,x,y,wall … name,x,y,wall**

For example, three users (Jack, Jill, Tron) are playing a game. The server program broadcast message might look like this:

**Jack,10,10,on Jill,12,10,off Tron,10,14,on Jack,11,10,off Jill,12,11,off Tron,10,13,on Jack,12,10,off Jill,12,12,on Tron,10,12,on Jack,13,10,on Jill,12,13,on Tron,10,11,on Jack,14,10,on Jill,12,14,on Tron,10,10,on**

Records are separated by a single space character. Each record contains the name of a user, the position of their light cycle on the grid, and jet-wall status of their light cycle.

When the user interacts with their light cycle the client program sends a message to the server. After a client sends a request, it must wait for the server to respond. The server should respond immediately to each client request.

The client requests and corresponding server responses are:

|  |  |
| --- | --- |
| Client request | Server response |
| ADD USER name CYCLE COLOUR value | OKAY | FAILED reason |
| GAME STATE | IDLE | WAITING | PLAYING |  GAME OVER winner |
| GET LEADERBOARD | OKAY name:score name:score … name:score |  FAILED reason |
| USER name TURN clockwise | anticlockwise | OKAY | FAILED reason |
| USER name GO faster | slower | OKAY | FAILED reason |
| USER name JETWALL on | off | OKAY | FAILED reason |

# Program Behaviour

The start-up sequence for the game is:

* The server program starts first:
  + The number of players is set between 3–10 for the new game
* Each client program starts next:
  + Users select unique names and light cycle colours

When the server receives an ADD USER request:

* If the user name is already taken an error message is returned and the request fails
* Otherwise, the server randomly assigns a position for the user’s light cycle on the grid, spaced apart from all other light cycles
* Each newly added light cycle must be facing in a direction that won’t mean the user immediate derezzes once the game starts

The GAME STATE request allows the client program to tell:

* When the game is playing, or
* If the server is waiting for more players to join, or
* If the game is over and the name of the winning user

The GET LEADERBOARD request allows a client program to display the leader board

* The server maintains the leader board in a text file

The USER requests allow the user to play the game

* If something goes wrong the server might return an error message

# Assessment Task 1 Specification and Planning

You are not expected to develop the 2D custom graphics necessary to display the game in this part of the project work. Instead, focus on the following aspects of the project:

1. Create a private GitHub repository for your project work and share it with your lecturer and your practical class supervisor
2. Create about 5 user stories (independent, valuable, estimable, small, and testable) for the various aspects of the game play requirements
3. Create “spike” user stories that allow you to explore how to use UDP multicast data packets to send and receive game state information based on the text-based data-protocol (see below)
4. Create a UML class diagram that represents the proposed structure of your project
5. In weekly iterations, incrementally commit production code along with corresponding tests based on your user stories and UML class diagram to your GitHub repository

As you implement your codebase make sure to:

* apply the Java coding style as supported by IntelliJ IDEA;
* add descriptive comments as necessary;
* use an appropriate naming convention for all identifiers (classes, variables, methods); and
* create classes and functions that each have a singular cohesive purpose.

When you implement your unit tests, base them on the acceptance tests of your user stories. Normally the “customer” is responsible for providing the priorities and acceptance tests for your user stories. However, you are required to take on this task.

You are expected to make reasonable progress towards implementing this codebase. It is okay if you don’t have all your user stories implemented and tested. Instead, aim to complete enough user stories to implement some of the core game logic and all the request/response data-protocol behaviour (see below).

It’s also okay to implement parts of your codebase in ways that might be improved on and even replaced as you learn new coding strategies during the semester. For example, you might start with manual testing of your code, and replace that with unit tests as you learn about them during the semester.

# Assessment Task 2 Specification and Planning

In the final part of the project work, you are expected to complete the network logic, game logic, and 2D custom graphics necessary to play the game. The requirements are:

1. Continue adding commits to the same GitHub repository
2. Continue improving your user stories and UML class diagram - they must always correspond to your project implementation
3. Include at least three (3) JUnit tests that verify the stability of the core parts of your game logic code
4. The server program remains a console application – light cycle position broadcasts must occur at a rate of 10 updates/sec
5. The client program becomes a Swing GUI – animations must update at a rate of 20 frames/sec
6. When the game starts, the minimum speed of a light cycle is 0 (pixels/frame), otherwise the speed is in the range 1-10 (pixels/frame)
7. Users must be able to direct their light cycles around the grid
8. If a light cycle hits the edge of the grid, then that user is automatically “derezzed”
9. The game allows a single user to win and have their score added to the leader board
10. Apply appropriate code readability and practices to your project work