“Statement of Work” for

CA314 Continual Assessment Team Project

0. Overview

Group Project: The continual assessment element of CA314 will be based on a single group project assignment to be carried out throughout the semester, with specified intermediate deliveries. The assigned project is a reasonably substantial one and is required to be done in teams. Normally, teams will consist of **6 students**. Departures from this team size will only be allowed for strong, valid reasons. The membership of each team should be notified to renaat.verbruggen@dcu.ie by 17.00 on Monday 16th September.

NOTE that, to work effectively, teams of this size must be well organised and business-like in their approach. Specifically
- It is required that each team hold regular, frequent meetings and records of these meetings are to be submitted as part of the deliverables for the assignment. [Use the Agile Scrum approach].
- Meeting records should at least indicate who attended, the current status of the team's work, any decisions made at the meeting, and plans to the next meeting.

All team members should contribute equally to the work but if this is not the case the team should inform renaat@computing.dcu.ie as to the approximate proportion of work contributed by each person.

Outline of Tasks: The tasks making up the assignment are detailed in Section 2, as well as the deliverable items and their deadlines for submission[i]. It is required that any UML modelling will be done with the aid of a software tool – *PlantUML for example* but a team may decide to use a different tool. The work is divided into four main elements, namely

Element	**Analysis**	**Product & Class Design**	**Implementation (Provisional+Final)**	**Presentation**	**Total C.A.**
Mark:	5%	7%	8%	5%	25%

Guidelines: Guidelines on the different elements of the work are presented in Section 3.

1. Functional Requirements: OPTIONS A & B

1.0 First team meeting, Preparation and Agenda

NB: This subsection, particularly, applies whether Option A or Option B is chosen.

(From Section 1.10 of /1/)
“ …
Each team member should read the following project requirements statement and create a list of questions and ambiguities that arise from it. The statement should be read multiple times until its basic functionality can be discussed without excessive referral to the written statement. After all team members have had the opportunity to familiarize themselves with the requirements, the team should meet to discuss the statement and address the following items:

- Identify and address questions and ambiguities
- Identify project elements requiring network programming (since this particular project is internet-based)
- Identify project elements requiring graphic image manipulation
- Identify project elements that lend themselves to audio
- Identify project elements requiring special algorithm creation
- Create a preliminary project conceptualization
- List three to six key classes comprising the system
- Establish areas of interest on which individual team members would like to work

[refer also to sections 2, 3 and 4 of this “statement of work”]

1.1 Project Overview

1.1.A Project Overview – Option A

The goal of the assignment is to implement an Internet-based, strategy game, called MyChess, which is closely modelled after the board game Chess. The game is Internet-based; that is, players will be able to play each other from remote sites on the Internet. This application will, therefore, take on a client/server structure, where each client supports one player’s view of the game while the server coordinates communication between players and orchestrates the sequence of events of the game.
The game description and background is here:

https://en.wikipedia.org/wiki/Chess

The goal is to produce a game that is exciting and likely to hold players’ interest. It is expected that teams will come up with their own variations of rules and moves in order to achieve this. For example, one might consider having an option of playing against the computer.

1.1.B Project Overview – Option B

The goal of the assignment is to implement an Internet-based game, called MyBridge. The game is Internet-based; that is, players will be able to play each other from remote sites on the Internet. This application will, therefore, take on a client/server structure, where each client supports one player’s view of the game while the server coordinates communication between players and orchestrates the sequence of events of the game.

The details are here:

https://en.wikipedia.org/wiki/Bridge

The goal is to produce a game that is exciting and likely to hold players' interest. For this option, it is expected that teams will come up with their own variations of rules and moves in order to achieve this. For example, one might consider having an option of playing against the computer.

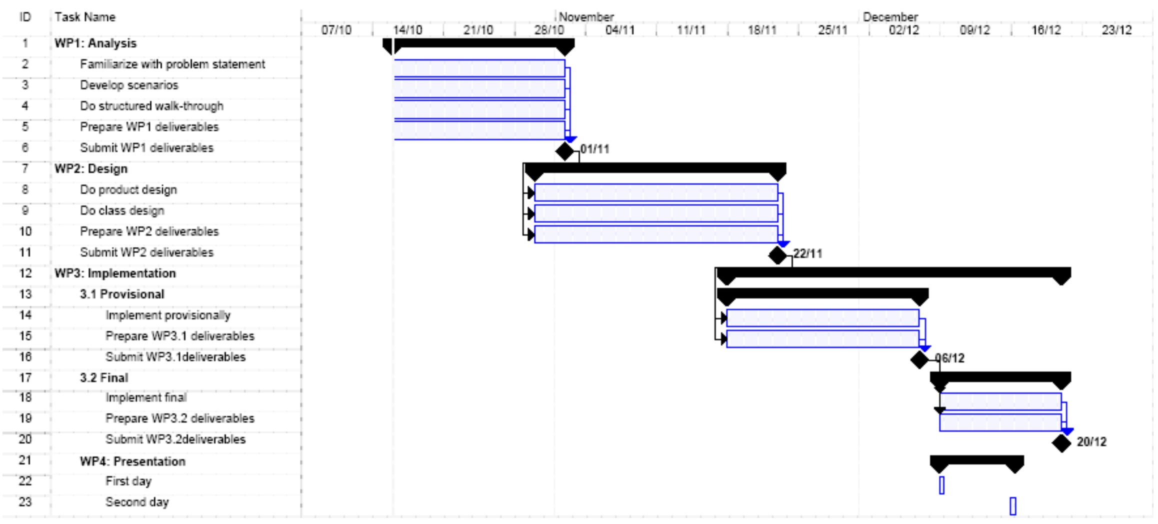
2. Assignment Deliverables & STRICT Due dates

Devel, Phase	**Deliverable**	**Due Date**	
Analysis *(to week 5)* **(see §3.1.1,§3.1.2)**	1. Refined requirements specification	**Fri. Oct. 4th, at 17.00** (End of week 4)	
	2. Scenarios		
	3. Primary class list		
	4. Class diagrams		
	5. Use case diagrams		
	6. Result of "Structured walk-through"[1]		
	7. Minutes/notes of team meetings		
Product Design *(weeks 6, 7)* **(see §3.2.1,§3.2.2)**	1. Object diagrams	*Should aim to have these parts done by Fri. Oct18th (end of week 6)*	**Fri. Oct. 25th,**
	2. Refined class diagrams		
	3. User interface mock-ups		
	4. State machines (***see section 3.2.1***)		
Class Design (*mainly weeks 6, 7 – maybe some refinement in week 8*) **(see §3.2.3)**	1. Collaboration diagrams		
	2. Sequence diagrams		
	3. Object diagrams		
	4. Refined class diagrams	**at 17.00** (End of week 7)	
	5. Class skeletons		
	6. Minutes or notes of any team meetings		
Implementation (provisional) *(weeks 8, 9, 10)* **(see §3.3)**	1. Source code (provisional)[ii]	**Fri. Nov. 15th, at 17.00** (End of week 10)	
	2. Minutes or notes of any team meetings		
Presentation *(weeks 11, 12)* **(see §3.4, §3.5)**	1. Project presentation	**Nov. 18th/Nov 21th/ Nov 25th 29th (in lecture/tutorial slots)**	
Implementation (final) *(weeks 11, 12)*	1. Summary of changes and additions since provisional source code delivery of Nov.28th. 2. Minutes or notes of any team meetings	**Fri. Nov 29th at 17.00** (End of week 12)	

Figure 2-1: Deliverables and Schedule

[1] The structured walkthrough is likely to result in some modifications of the other deliverables. It is fine to just note this in meeting minutes, for example.

The following figure (Figure 2-2), prepared using Microsoft Project, is a Gantt chart that provides a summary of the overall project schedule.



ID	Task Name
1	**WP1: Analysis**
2	Familiarize with problem statement
3	Develop scenarios
4	Do structured walk-through
5	Prepare WP1 deliverables
6	Submit WP1 deliverables
7	**WP2: Design**
8	Do product design
9	Do class design
10	Prepare WP2 deliverables
11	Submit WP2 deliverables
12	**WP3: Implementation**
13	**3.1 Provisional**
14	Implement provisionally
15	Prepare WP3.1 deliverables
16	Submit WP3.1deliverables
17	**3.2 Final**
18	Implement final
19	Prepare WP3.2 deliverables
20	Submit WP3.2deliverables
21	**WP4: Presentation**
22	First day
23	Second day

Figure 2-2: Summary Timeline for Project Development

3. Some guidelines for activities & deliverables

3.1 Analysis phase

3.1.1 Familiarizing with Problem/ Creating informal scenarios

(Based on section 2.2 of /1/)
"… The first step is to read the requirements specification provided in section 1.
…
Once the requirements specification (Section 1) has been read, students should begin to create **informal scenarios** based on the specification. *It might help for teams to play their chosen game to make sure that everyone understands what is involved – this has been useful for teams in the past.* Through an informal scenario, we tell a "story" of how users will be using the system. The story should be concrete. For example, a poorly written informal scenario tells a story of "user A entering all necessary data in the dialogue window" whereas a well-written informal scenario specifies that "Andrea enters her name and password in the login screen".

3.1.1.1 Guidelines for Creating Informal Scenarios {USER STORIES}

"…

- There should be a number of small informal scenarios, rather than one large informal scenario.
- An informal scenario should address one coherent aspect of the system, such as *user logs on to system, server deals cards to users, user makes a move, user wins/loses the game*, and so on.
- Each informal scenario should specify concrete values whenever possible. Rather than writing that *a user spins the spinner and moves the specified number of spaces*, students should write that *Andrea spins the spinner, which lands on the number 6. She*

then moves four spaces forward and two spaces left. The latter conveys the potential complexity of managing user moves on the game board, while the former does not.

• In the informal scenarios, you should address some types of user error that the system will handle, but you should not attempt to exhaustively cover all possible errors.

• Implementation details should be omitted in expressing each informal scenario. For example, an informal scenario should not mention linked lists or other data structures.

• Each scenario should describe the state of the system upon initiation of that scenario. For example, the *user makes a move* informal scenario should describe the example configuration of the game board prior to the user move.

• Each scenario should terminate by indicating which scenario is next.

The tendency is to make the informal scenarios too abstract rather than sufficiently concrete. The problem with characterizing the use of the system in abstract terms is that the complexities of the system are not apparent when described abstractly as when a detailed situation is discussed. The point of creating informal scenarios is to allow the developers to gain increased understanding of the project to be developed."

3.1.1.2 Sample Informal Scenario: User Makes a Move

"Current System State
The system state consists of each player at his or her starting location in the game board. The three players in the game, Andrea, Max, and Emma, have each been dealt six cards. Because the value of each card is irrelevant to this informal scenario, these values are omitted.
Informal Scenario
Andrea has the next move. She spins the spinner, which lands on the number 5. Andrea has the white playing piece. She moves this piece one space to the left, one space towards the top of the game board, two spaces to the right, and finally, one space to the top of the game board. Because of the final position of her game piece, Andrea has no additional options, and her turn ends.
Next Scenario
This scenario is repeated with the player to the left of Andrea. Therefore, Max has the next turn."

NB: "Break the playing of the game (for the assignment) into as many different informal scenarios as you can think of. Divide these scenarios among your project development team members. Each of you should come up with informal scenario descriptions using the guidelines of section 3.1.1.1."

3.1.2 Analyzing the project

(Section 3.11 of /1/)
The requirements statement provided in section 1 is the ***requirements specification*** for the assignment. Normally, developers can consult domain experts to get clarification of the requirements – in this case, the domain expert role will have to be played by the lecturer. However, it is likely that a good many people in the class are familiar with the game which should be very helpful towards understanding the requirements.

Each team should have created a list of questions from their first reading of the requirements specification. After these questions have been refined by team discussion, any unanswered ones should be addressed to the module lecturer. The resolution of these questions should be documented (e.g. in a team's log of the project). After gaining familiarity with the project, including through creation of informal scenarios, and after resolving any open questions, each team should continue the analysis process through the following sequence of steps.

- A list of primary classes should be created and refined ….
- A set of basic class diagrams should be developed using these classes. The class diagrams should show aggregation and inheritance [generalization]
- Use cases should be developed concerning the major functionalities of the system.
- Class diagrams modelling the use cases should be developed.
- Use case diagram(s), in which the use cases are shown in relationship to each other, should be created.
- Scenarios relating to each use case should be created.
- Each team should engage in a structured walk-through … to verify the completeness and correctness of the proposed system [and its verifiability]..

[So] deliverables resulting from the analysis phase are the refined requirement specification [essentially questions raised & their resolution including any resulting changes in the original specification], use cases, scenarios, class diagrams, and use case diagrams. Each project team should discuss any technical vulnerabilities.

3.2 Product and class design phases

3.2.1 Inter-process communication in the project [client/server etc]

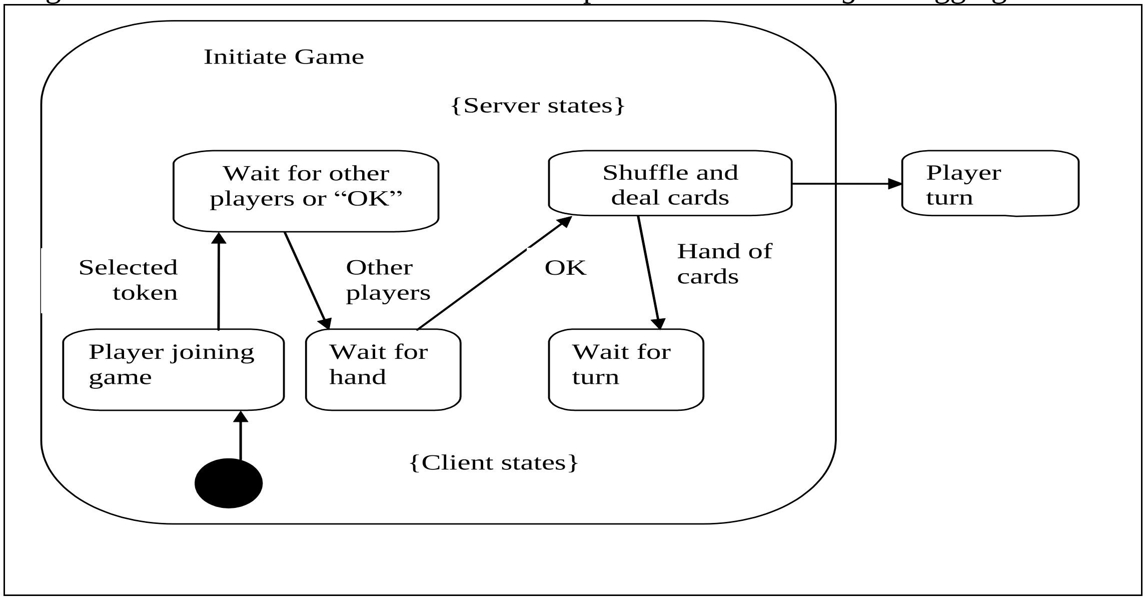
(Section 4.9 of /1/)

"One of the objectives of the class project is to co-ordinate the playing of the game between remote players over the internet. Accordingly, this system takes on a **client/server** architecture. Each client represents a remote player, and the server coordinates the game playing among the players. [A UML deployment diagram could be used to depict this]. The classic client/server system partitioning involves the server process disseminating centralised information to the client processes, and each client process creating the user interface for its client use. Beyond that intuitive division of responsibility in client/server systems, determining the comprehensive functionality of the client versus the server is more challenging.

For example the software developed for the class project must embody the rules for the game. Does each client process individually enforce these rules. Or should that logic reside on the server? Either scenario seems reasonable, and there are many such decisions that the software designer must make. For example, each client process can determine which player takes the next turn, or the server can execute the sequence of player turns.

A very basic rule for dividing tasks among clients and a server involves determining the number of clients that would need to execute a given set of logic at a given time. That is, in the case of determining which player should take the next turn, if the logic resides on the client, then every client process must execute that logic at the end of every turn. This logic, therefore, is an excellent candidate for residing on the server. However, the logic embodying the rules of the game is a good candidate for residing on the client since only the client whose player is currently taking a turn would need to execute this logic at a particular time.

After the responsibility of game playing has been distributed among the client and server classes, the communication between these components must be designed. Figure 3.2.1-1 shows a state machine that represents the *initiate game* aggregate state.



<u>Figure 3.2.1-1: State Machine for Initiating a Game [client-server comm.]</u>

The initial state, represented by the black dot, unconditionally transitions into the *player joining game* sub state (more detail given in Figure 3.2.1-2). Once the *player joining game* state has executed, the token selected by this new player is communicated to the server sub state called *wait for other players or "OK"*. The transitions between sub states represent information passed between client and server processes over a socket connection.

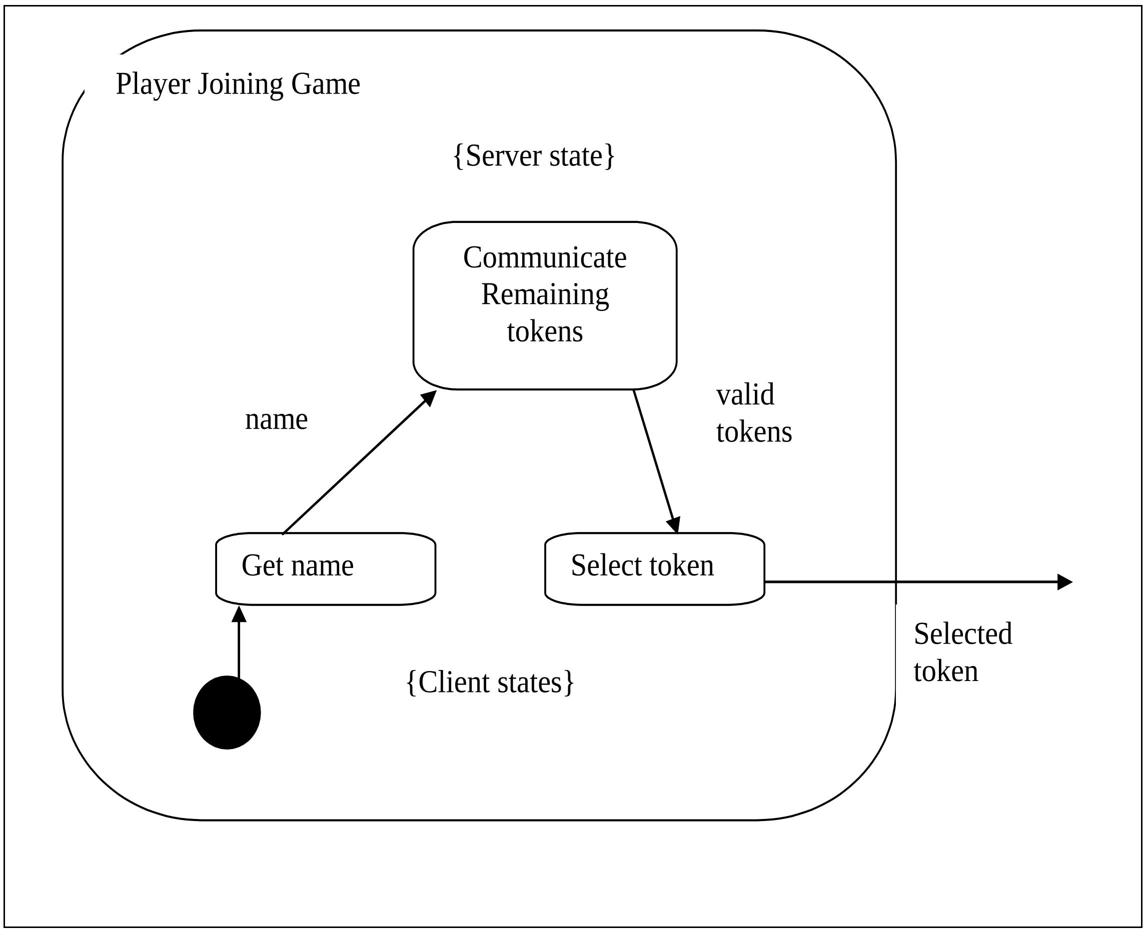


Figure 3.2.1-2: State Machine for Player Joining a Game

Looking at Figures 3.2.1-1 and 3.2.1-2, the *initiate game* state begins at the *player joining game* sub-state, which in turn begins at the *get name* sub-state, part of the client. This sub-state then communicates the player name to the server, and a transition is made to the *Communicate remaining tokens* sub-state, which sends a list of valid tokens to the client. The transition is then made to the *select token* state, which communicates the user-selected tokens to the server. The transition is then made from the *player joining game* state to the *wait for other players or "OK"* state, which waits for additional players to identify themselves. Once at least three players are ready to play the game, the first player who logged on may start the game at any time by selecting an OK button. As players log on to the system, their presence is communicated to each client. Once sufficient players are logged on and the first player initiates play, the *Shuffle and deal cards* sub-state is This sub-state communicates the hand of each player to that player, and then unconditionally enters the *player turn* state. The *player turn* state may be broken down similarly to the way *initiate game* sub-state has been here.

Figure 3.2.1-3 shows the entire Galaxy Sleuth inter-process communication. Notice that the *initiate game* state shown in detail in Figure 3.2.1-1 is placed in context in Figure 3.2.1-3. The *initiate game* state is entered unconditionally from the initial state. The transition to the next state, *player turn*, is unconditional. Which state is entered next depends upon the action taken by the player on his or her turn. If the player makes no hypothesis or conclusion, the state machine enters the *player turn* state again (for the next player, of course). If, however, the player formulates a hypothesis, the state machine enters the *attempt to refute hypothesis* state and then unconditionally returns to the *player turn* state. Finally, if the player formulates a conclusion, the state machine enters the *verify murder scenario* state. If the user has guessed incorrectly, the state machine enters the *remove player* state and then returns unconditionally to the *player turn* state. If the user has guessed correctly, the game is over, and the state machine enters the final state.

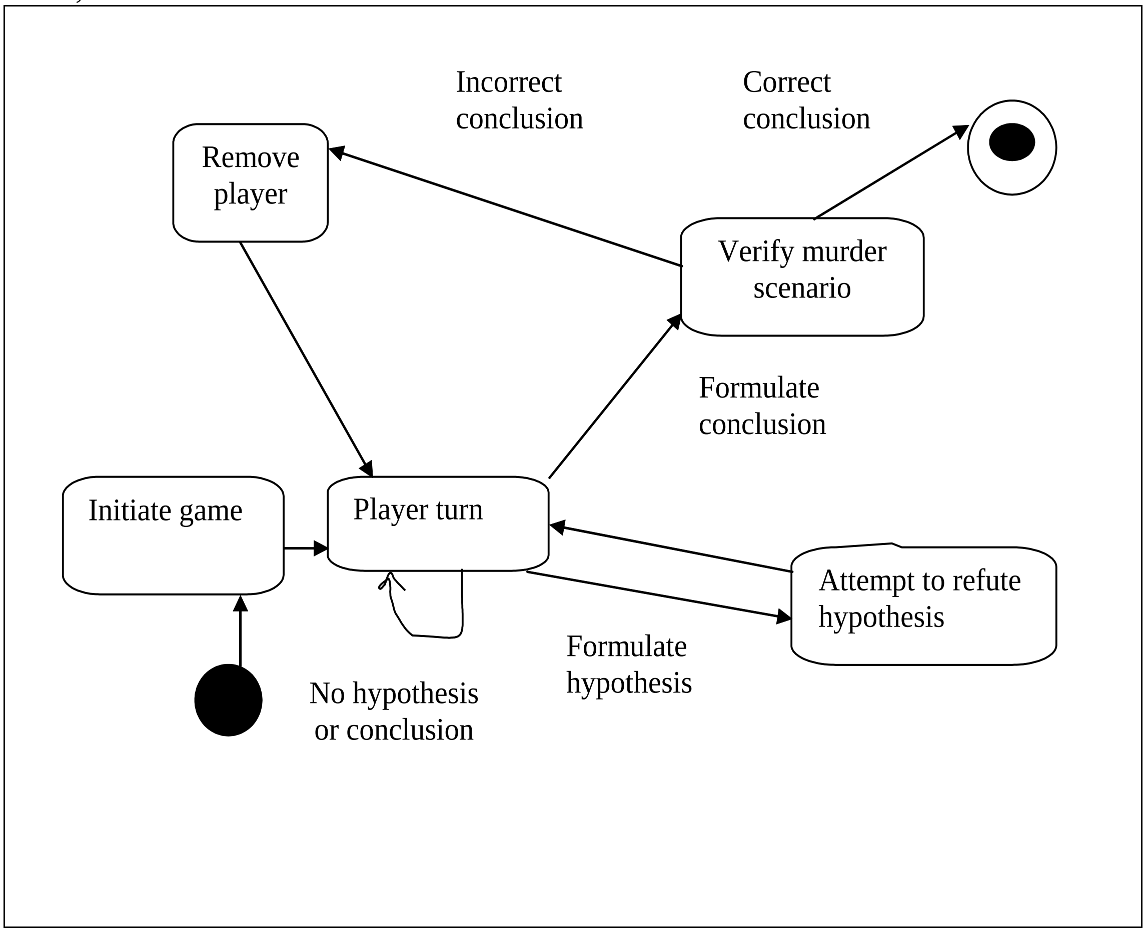


Figure 3.2.1-3: High-Level State Machine Representing Galaxy Sleuth

Figure 3.2.1-3 consists of very abstract states, each of which may be broken down into its respective sub-states as has been done with the *initiate game* sub-state. Some of the sub-states shown in Figure 3.2.1-1 may themselves be deemed to be too abstract, and may be broken down into their sub-states as has been done with the *player joining game* state. In fact, an entire hierarchy of state machines may be created until only very self-explanatory states are specified as building blocks of more complex states.

Figure 3.2.1-4 shows a partial hierarchy of state machines, where the parent node in the hierarchy is the state machine that is made up of sub-states that are represented as child nodes in the hierarchy. The **root** node represents the system as a whole and consists of the set of sets expressed at its most abstract level. In this case, each state merits decomposition into sub-states, but in the illustration only one of its second level states is decomposed.

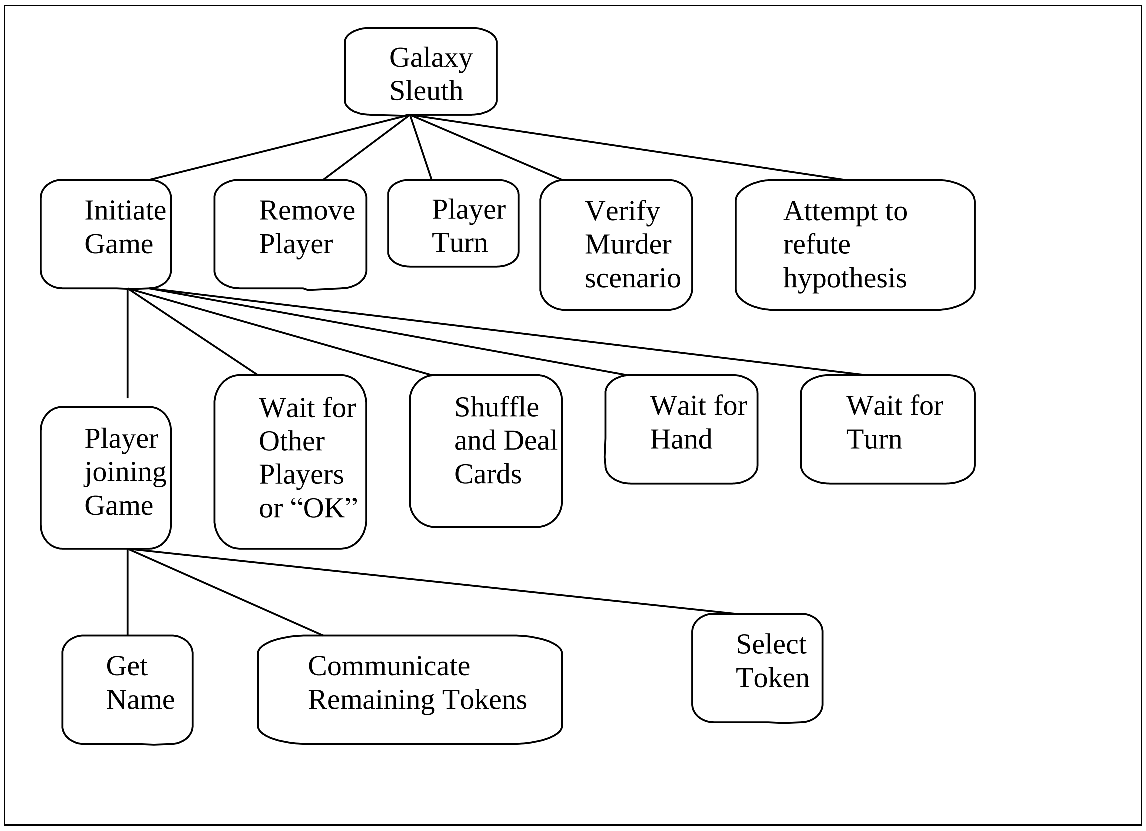


Figure 3.2.1-4: A Hierarchy of State Machines for Galaxy Sleuth [partial]

N.B.: The state machine ***deliverables*** for the assignment should be obtained by decomposing each of the states in Figure 3.2.1-4 until the label on the state is self-explanatory.

3.2.2 Assignment Project Product design

(Section 4.15 of /1/)
"The elements of product design are:
- Object persistence
- Process architecture
- User interface design
- Resource distribution
- Network utilization

The requirements specification for Galaxy Sleuth provides an illustration of one user interface, specifically the game board to be seen by the players, so all other interfaces must be designed by the system developers. [Similarly, the one user interface in the Adaptive Snakes & Ladders game board is to be seen by all players.] Because the assignment project is an example of a client/server system, the network-based communication between the clients and the server constitutes an essential aspect of the software structure. In addition, the client/server interaction also determines several interfaces for the client. The server does not directly interact with human users, and thus it requires no human-computer interfaces to be designed.

The steps for designing the user interface for Galaxy Sleuth are these:
- Storyboard the primary functionality of the system

• Determine which objects of the system require a graphic representation, and design these. For example, *Card* objects require such a representation.
• Add the secondary functionality of the system to the storyboard.
• Create mock-up (prototype) interfaces
• Get user feedback
• Refines interfaces"
Very similar steps apply for the Adaptive Snakes & Ladders user interface.

3.2.3 Designing the Project

(Section 5.6 of /1/)
"At this point in the software development process, all members of the development team should have clearly delineated responsibility for developing the assignment project. The delineation is most effective if it is along class boundaries so that development responsibility is assigned in terms of classes. Each team member should have responsibility for classes that constitute a functional partition of the project. Of course, each functional partition will require services from classes being developed by other team members. This interaction among functional partitions means that it is essential that development team members be in close communication during this period, so that each may express his or her needs to the appropriate team mate.

The primary design process for the project is to further develop models of the system behaviour so that definitions for the classes comprising the system can evolve. Further refining the class diagrams and creating collaboration, sequence, and object[iii] diagrams will establish a more complete portrayal of the attributes and methods necessary for each class. Each development team member must independently develop the diagrams necessary for his or her functional partition of the project.

There is a [good deal] of work to do during the design phase. … Given that the classes designed must interoperate effectively, one must accommodate class design revision during this … period. To allow for class refinement, team members should try to complete a preliminary design in time for an informal walkthrough … [early enough to allow] for revisions made apparent during the walk-through.

The design period addressed here encompasses both class design and product design. Thus, the graphical user interfaces, the game board, and the client/server architecture need to be designed … as well. These responsibilities can be distributed to development team members who have smaller functional partitions and thus fewer or less complex classes to develop.

The steps for completing class design are as follows:
• Review the set of scenarios to determine whether any additional scenarios need to be created to model significantly different aspects of any use cases.
• For new scenarios, create a class diagram showing the necessary interclass relationships for each scenario.
• For each scenario, consult its associated class diagram and develop either a collaboration or sequence diagram.
• Update the class diagrams of the classes that require additional attributes or methods to accommodate the inter-object links portrayed in the collaboration or sequence diagrams.

• Translate the resulting class diagrams into class skeletons[iv] to begin the implementation phase."

3.3 Implementation phase

(Section 7.9 of /1/)
"[Each team should define] programming standards for [their assignment]. Responsibility for class design has already been assigned to individual members of the project team. Ideally, all students should implement the classes they designed but … [some] adjustments may need to be made. Assign responsibility for implementing classes in a manner that maximises everyone's ability to code and test classes independently and that reasonably distributes the workload. Note that formal unit and integration testing is not required for this assignment but it is the responsibility of each coder to make sure that their code runs correctly.

[A good (*suggested*)] approach to implementation is the threads approach. … divide the project into partitions of functionality according to the use cases created during analysis. …"

The implementation is to have a provisional phase, prior to project presentations, and a final phase where loose ends are tied up, errors corrected, and perhaps unfinished parts are completed.

3.4 Completing & Presenting the Assignment

(Chapter 12 of /1/)
Each team must present their completed project to the full CA314 class. While teams are free to decide on the format of their presentation, it is important that all members of the team make some contribution.

Apart from a demonstration of the software in use, it is suggested that the presentation be a technical one. Thus, a good approach might be for the presentation to include the "characteristics:

- It should be structured around use cases
- It should illustrate the underlying class structure that supports each area of functionality
- It should highlight challenging algorithms, code re-use, and inter-process communication mechanisms."

In addition, it would be interesting to have an indication of any problems encountered, lessons learned, etc.

4. References

/1/ Stiller, E., LeBlanc, C., "Project-based Software Engineering", Addison-Wesley, 2002
/2/ Hunicke, R., LeBlanc, M., Zubek, R., "MDA: A Formal Approach to Game Design and Game Research, http://www.cs.northwestern.edu/~hunicke/MDA.pdf, 2004.

[i] All projects will be uploaded on Loop.
[ii] It is acceptable that the delivered source code might not be absolutely complete. A note listing any gaps should be included.

[iii] "An **object diagram** models a set of objects and their interrelationships during a system snapshot. A system **snapshot** is the state of the software system at a selected moment of time. The notational elements that make up object diagrams are objects and links. An object diagram models the states and interconnections of objects and serves as another static view of the system. Unlike other diagram types encountered thus far, an object diagram may contain multiple instances of a particular class. The representation of multiple objects shows that more than one instance of a class may be allocated at a particular point during execution."

[iv] "A [complete] class skeleton includes the following items:

- A list of the **roles** that the class plays within the system.
- Information concerning when objects of the class are created and deleted.
- For each role, the **semantics** of the class.
- All attributes, including access modifiers, types, names, and semantics (if the attribute name is not self-documenting), that constitute the class.
- All constructors with pre-conditions and post-conditions.
- For each method, its signature, semantics, pre-conditions, and post-conditions."