

“Statement of Work” for

CA314 Continuous Assessment Assignment (‘09-‘10)

0. Overview.....	2
1. Functional Requirements for the Assignment.....	3
1.1 Project Overview	3
1.2 Game Elements	4
1.3 The Game Sequence of Events	5
1.4 Moving and Landing on Planets	6
1.5 Winning the Game	7
2. Assignment Deliverables & “STRICT” Due dates.....	7
3. Some guidelines for activities & deliverables	8
3.1 Analysis phase	8
3.1.1 Familiarizing with Problem/ Creating informal scenarios.....	8
3.1.1.1 Guidelines for Creating Informal Scenarios	8
3.1.1.2 Sample Informal Scenario: User Makes a Move	9
3.1.2 Analyzing the project.....	9
3.2 Product and class design phases	10
3.2.1 Interprocess communication in the project [client/server etc]	10
3.2.2 Assignment Project Product design	14
3.2.3 Designing the Assignment	15
3.3 Implementation phase	16
3.4 System testing phase	16
3.5 Completing & Presenting the Assignment.....	16
4. References.....	16

0. Overview

The continuous assessment element of CA314 in 2009-2010 will be based on a single project assignment to be carried out and reported on throughout the semester.

The assigned project is a reasonably substantial one and, as such, is expected to be done (thoroughly) in teams. Normally, teams will consist of **6 students** and each team must have members from both CAIS and CASE. **In fact, given the composition of this year's class, there should be at least 2 CAIS and 3 CASE students in each group.** Departures from this type of team will only be allowed for strong, valid reasons. The membership of each team should be notified to ltoohy@computing.dcu.ie by 17.00 on Friday Oct 16th, 2009.

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.
- Records should at least summarise the current status of the team's work, any decisions made at the meeting, and plans to the next meeting.
- It is also required that each student writes an engineering diary to record day-to-day technical activities on the project, especially key problems arising and decisions taken.
- From time to time, the lecturer will meet each team separately to assess progress

It is expected that all team members will contribute equally to the work of the team but if this is not the case the team should inform ltoohy@computing.dcu.ie as to the approximate proportion of work contributed by each personⁱ.

The **functional requirements** for the project, taken from /1/, are presented in Section 1. Several copies of /1/ are available in the library, although for the purpose of the assignment the contents of this document are adequate. Still, /1/ is highly recommended for its insights.

The tasks making up the assignment are detailed in Section 2, as well as the deliverable items and their deadlines for submissionⁱⁱ. It is expected that any required UML modelling will be done with the aid of a software tool – there are two such tools on the lab machines, *WithClass* and *Rational Rose*. The work is divided into four main elements, namely

Element	Analysis	Product & Class Design	Implementen-tation	System testing & Presentation	Total C.A.
Mark:	5%	7%	8%	5%	25%

Guidelines, from /1/, on the different elements of the work, are presented in Section 3. Quite detailed guidance, including some of the actual work (!), is given for the Analysis phase and for various aspects of the Product & Class Design phase. These *should enable teams to get off to a good start* to the assignment.

1. Functional Requirements for the Assignment

(Section 1.10 of /1/ - renumbered here as Section 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

The goal of the assignment is to implement an Internet-based, whodunit-style game, called Galaxy Sleuth, which is closely modeled after the board game Clue [or Cluedo]. 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 is an intergalactic murder mystery that emulates a board game scenario in which players spin a wheel to determine a randomly selected number of moves by which they travel through space to various planets. The objective of the game is to gather a series of clues while traveling about the universe. Each player receives information concerning the murder, and no players receive the same piece of information. As the players travel through the universe, they are permitted to ask questions of their fellow players. The questions take the form of hypotheses about the circumstances and perpetrator of the murder, which the remaining players are invited to disprove with their evidence. The murder hypothesis is communicated to all players. If a player can disprove the hypothesis, that piece of information is communicated only to the inquiring player. The other players are informed that the hypothesis was refuted, but are not shown the piece of evidence that refuted the hypothesis. If the hypothesis cannot be refuted, all players are informed of that fact. After gathering a certain set of clues and applying deductive reasoning, players may deduce the perpetrator and circumstances of the murder. The first player to accomplish this goal wins the game. If a player draws an incorrect conclusion, he or she loses the game.

The game involves elements of both luck and strategy. The objective of player movement throughout the universe is to investigate specific planets to determine whether each was the murder site. Part of the game, therefore, involves reaching suspicious locations quickly to investigate them. This element is dictated largely by luck, since the spin of the wheel determines the speed at which a player reaches a destination. Players may also travel through a few strategically placed wormholes. The wormholes connect certain planets to each other. The strategy of the game is embodied in selecting questions to ask other players, selecting planet destinations, and efficiently deducing the correct murder scenario.

1.2 Game Elements

The intergalactic context for the game consists of nine planets, *Cathitar*, *Earth*, *Evilon*, *Linuta*, *Pheadrum*, *Ping*, *Psu* and *Verlute*. Three to six players may participate in this game. Each player has a graphic representation of the galaxy on or her remote computer. Each player is represented by a token of a different colour in the game window. Each token represents a character in the murder mystery. Each player is, therefore, both trying to solve the murder mystery and taking on the identity of one of the six murder suspects. The murder suspects are *Tina Time Traveler* (red token), *Sam Space Voyager* (blue token), *Mona Moon Walker* (yellow token), *George Galaxy Wanderer* (green token), *Uhura Universalist* (purple token), and *Steve Stargazer* (white token).

Each token occupies a physical location in the galaxy at any given time. When a player (as represented by the token) is not residing on a planet, his or her token may reside on one of the locations in space. Space is segmented into squares as shown in Figure 1.2-1, the graphic representation of the galaxy. To move between planets, a player may either spin the spinner in the centre of the game by clicking on it or select a wormhole. The spinner will land on a randomly selected number between one and six. In order to move through a wormhole, the player must be residing on one of the four planets which has access to one.

The game environment also contains six lethal instruments: a *phaser*, a *hyper rope*, a *laser sword*, *biological agents*, a *flamethrower*, and *radioactive chemicals*. One of these items is the murder weapon. The final element of the game is a window that may be used by players to record the results of the questions posed to the other players. This window serves as an aid in deducing the circumstance and perpetrator of the murder.

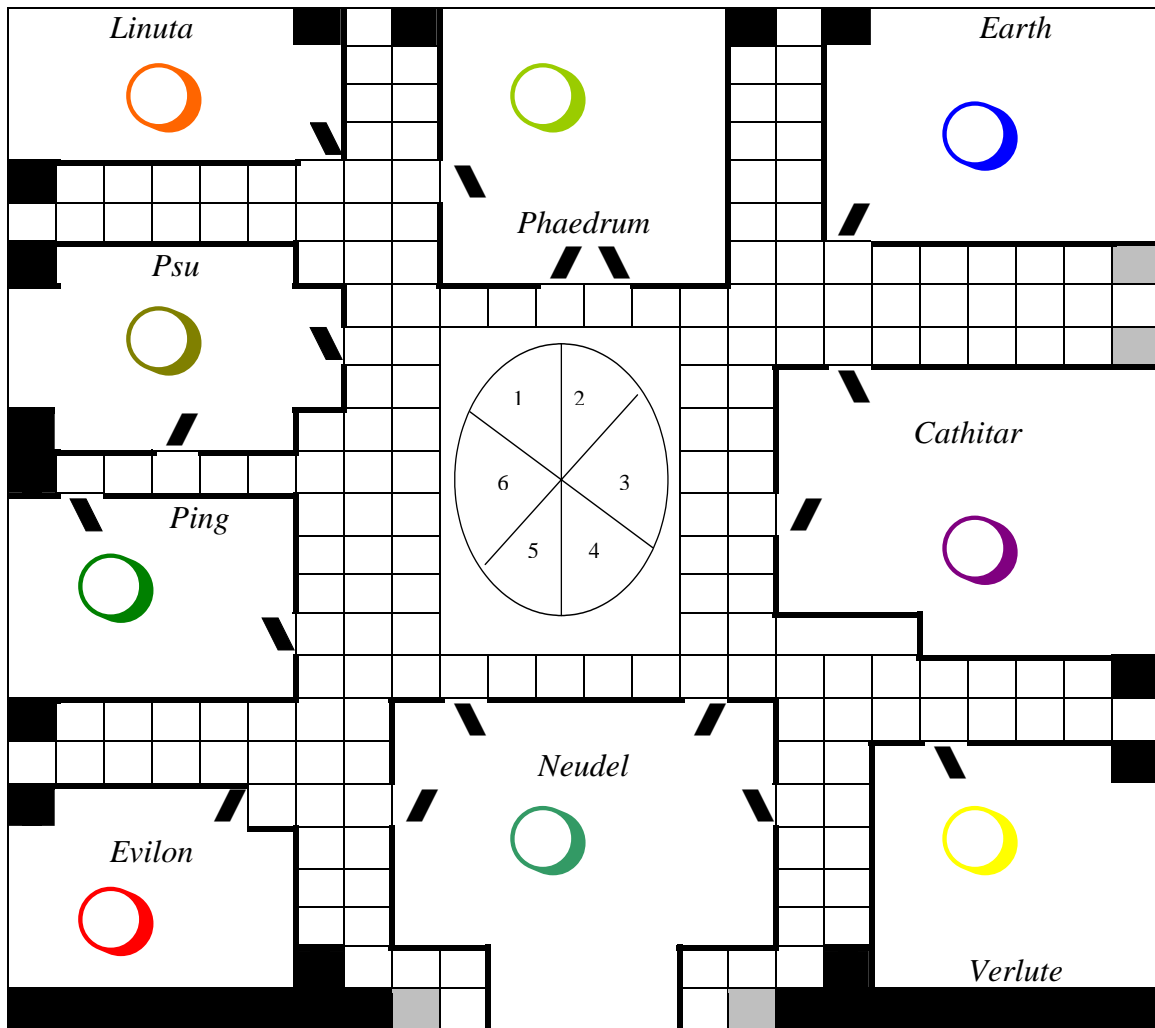


Figure 1.2-1: A Depiction of the Game Board

1.3 The Game Sequence of Events

Players of Galaxy Sleuth point their web browsers to the home page containing the client applet code or invoke a local version of the client. Which action the players take depends on the language used to implement the assignment. The player selects either a *start game* or *join button* depending whether a game has already begun or not. The first player to initiate a game decides on the parameters for waiting for additional players to join the game and may select a suspect and associated token to move during the game. The player either selects a time limit for when to begin the game or chooses to wait for a specified number of additional players. As players join the game, each selects a game token from those remaining. Once sufficient players join the game or the time limit elapses with at least three players, the game begins.

First, the game server selects a random murder scenario, consisting of a perpetrator, a murder weapon, and a planet where the murder takes place. The remaining evidence consists of the suspects, excluding the actual murderer, the possible murder weapons, excluding the actual murder weapon, and the planets, excluding the site of the murder. This evidence is then distributed to the players one piece at a time, starting with the first player who joined the game and continuing in order of player sign-on until the evidence is exhausted. If a player has a piece of evidence, for example the planet

Earth, that player knows the murder did not take place on *Earth*. Therefore, all the evidence held by a player serves to refute a part of a murder hypothesis.

Each suspect and associated token has a fixed starting point in the game represented by a specific cell in the game representation. The game-board representation places each token in its respective starting location. The predetermined playing order starts with *Tina Time Traveler*, and thus the player who selected the associated token moves first. The remainder of the order is *Sam Space Voyager*, *Mona Moon Walker*, *George Galaxy Wanderer*, *Uhura Universalist*, and *Steve Stargazer*.

A player turn consists of a spin of the spinner followed by the movement of the player's token the appropriate number of cells or the movement of the player's token through a wormhole (if the player's token is on one of four special planets with a wormhole connection to another planet). If the player lands on another planet as a result of the move, he or she may pose a murder hypothesis to the following player. In other words, the first player poses his or her question to the second player joining the game, the second queries the third, and so on. The last person queries the first player. If the player to whom the hypothesis was posed cannot refute the murder hypothesis with a piece of evidence, then the next person in the starting sequence is asked, until all the players are exhausted or someone successfully refutes the hypothesis. To refute a hypothesis, a player shows only a single piece of evidence even if the player holds more than one piece that refutes the hypothesis. Each player also has a special window to assist in keeping track of the evidence shown and hypothesis outcomes.

If a player attempts to respond erroneously to a murder hypothesis, for example, by responding that a hypothesis cannot be refuted when he or she has a refuting piece of evidence or by trying to show an irrelevant piece of evidence, the game should prohibit this attempt from succeeding. In other words, the game allows only correct responses to be given, but if there is more than one correct response, the responding player can choose which he or she would like to give.

1.4 Moving and Landing on Planets

In order to visit a planet, a player must enter the planet's atmosphere at one of the predefined entry points. Each planet has between one and three predefined atmospheric entry points. These entry points are graphically depicted on the game representation as a slanted line off a square on the board leading into the atmosphere of a planet. [/ or \ in Figure 1.2-1]. A player visits a planet by making legal moves on the board until a cell adjacent to an atmosphere entry point is reached. It costs the player a single move to enter the atmosphere from the adjacent cell, and the player is considered to be on the planet as soon as the planet's atmosphere has been entered. The player does not have to enter the planet's atmosphere in an exact number of moves. A player whose token resides on one of the four planets having wormhole connections to other planets may choose not to spin the spinner and instead take the wormhole directly to the other planet. The planets *Linuta* and *Verlute* have a wormhole connecting each other, as do *Earth* and *Evilon*. A player may not revisit the same planet on a single turn.

A legal move consists of a player moving his or her token the number of moves determined by the spinner, either horizontally or vertically, but not diagonally. The player may change directions any number of times, but cannot enter the same cell

twice during a single turn. Also, a player may not move into a cell occupied by the token of another player.

1.5 Winning the Game

A player wins the game by announcing his or her murder hypothesis (as a solution to the murder mystery) consisting of a planet, a perpetrator, and a murder weapon during his or her turn, and having the server determine that the hypothesis is correct. The solution is then broadcast to all players. If an announced hypothesis is not correct, the player loses the game and cannot pose hypotheses any longer or make moves on the board, but must continue to refute the hypotheses of other players. When the correct hypothesis is announced all players are informed of the circumstances of the murder.”

2. Assignment Deliverables & “STRICT” Due dates

Devel, Phase	Deliverable	Due Date
Analysis (weeks 3, 4, 5) (see §3.1.1,§3.1.2)	1. Refined requirements specification	Fri. Oct. 30th, 2009 at 17.00 (End of week 5)
	2. Scenarios	
	3. Primary class list	
	4. Class diagrams	
	5. Use case diagrams	
	6. Structured walk-through	
	7. Minutes/notes of team meetings	
Product Design (weeks 6, 7) (see §3.2.1,§3.2.2)	1. Object diagrams	<i>Should aim to have done by Fri. Nov 13th(end of week 7)</i>
	2. Refined class diagrams	
	3. User interface mock-ups	
	4. State machines (<i>see section 3.2.1</i>)	
Class Design (mainly weeks 6, 7 – maybe some refinement in week 8) (see §3.2.3)	1. Collaboration diagrams	Fri. Nov. 20th, 2009 at 17.00 (End of week 8)
	2. Sequence diagrams	
	3. Object diagrams	
	4. Refined class diagrams	
	5. Class skeletons	
	6. Minutes or notes of any team meetings	
Implementation (weeks 8, 9, 10) (see §3.3)	1. Implementation plan	End of Week 7
	2. Source code ⁱⁱⁱ	Fri. Dec. 4th, 2009 at 17.00 (End of week 10)
	3. Minutes or notes of any team meetings	
System Testing & Presentation (weeks 11, 12) (see §3.4, §3.5)	1. System test plan	End of Week 7
	2. Project presentation	Tue Dec 15th, 2009
	2. System test analysis report	Fri. Dec. 18th, 2009 at 17.00 (End of week 12)
	4. Final system delivery	
	5. Minutes or notes of any team meetings + Individual engineering diaries	

Figure 2-1: Deliverables and Schedule

Week number	3	4	5	6	7	8	9	10	11	12
Activity										
Familiarize with problem statement	█									
Develop scenarios		█	█							
Create class diagrams			█							
Do structured walk-through				█						
Refine after walk-through					█					
Create collab. & seq. diagrams						█				
Create object diagrams							█			
Refine class diagrams								█		
Create class skeleton									█	
Do informal walkthrough/Refine										█
Perform programming										
Do system testing										
Refine and re-test										
Present in class										

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 has been read, students should begin to create **informal scenarios** based on the specification. 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

“... ”

- 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 formal 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 Clue or Cluedo 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 diagrams, 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] the formal 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 and bring these to the attention of the lecturer.”

3.2 Product and class design phases

3.2.1 Interprocess 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 Galaxy Sleuth. 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.

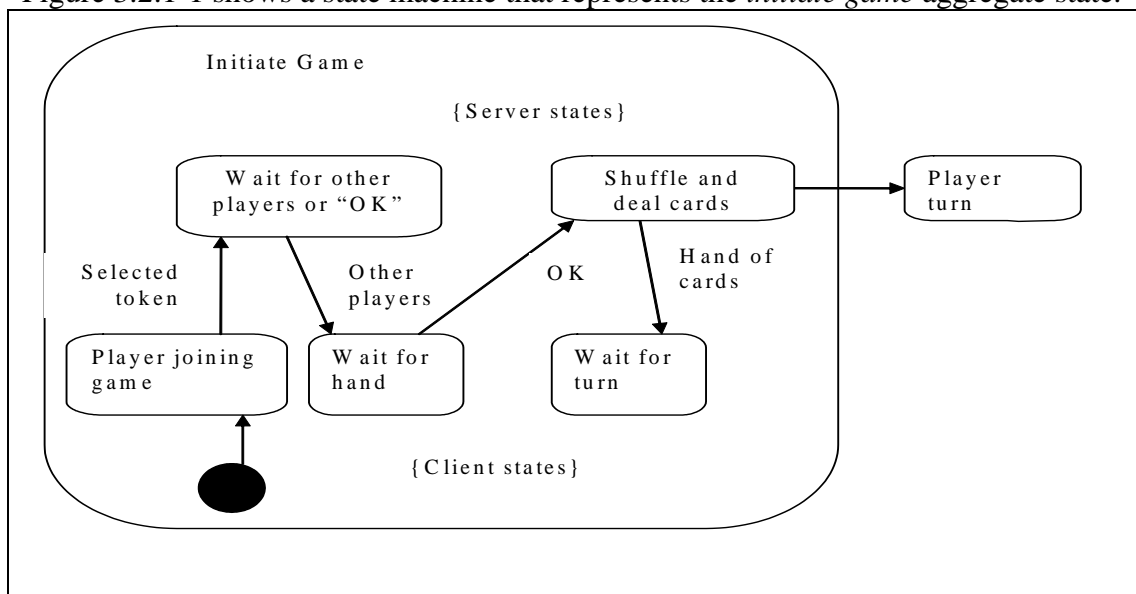


Figure 3.2.1-1: State Machine for Initiating a Game [client-server comm.]

The initial state, represented by the black dot, unconditionally transitions into the *player joining game* substate (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 substate called *wait for other players or "OK"*. The transitions between substates represent information passed between client and server processes over a socket connection.

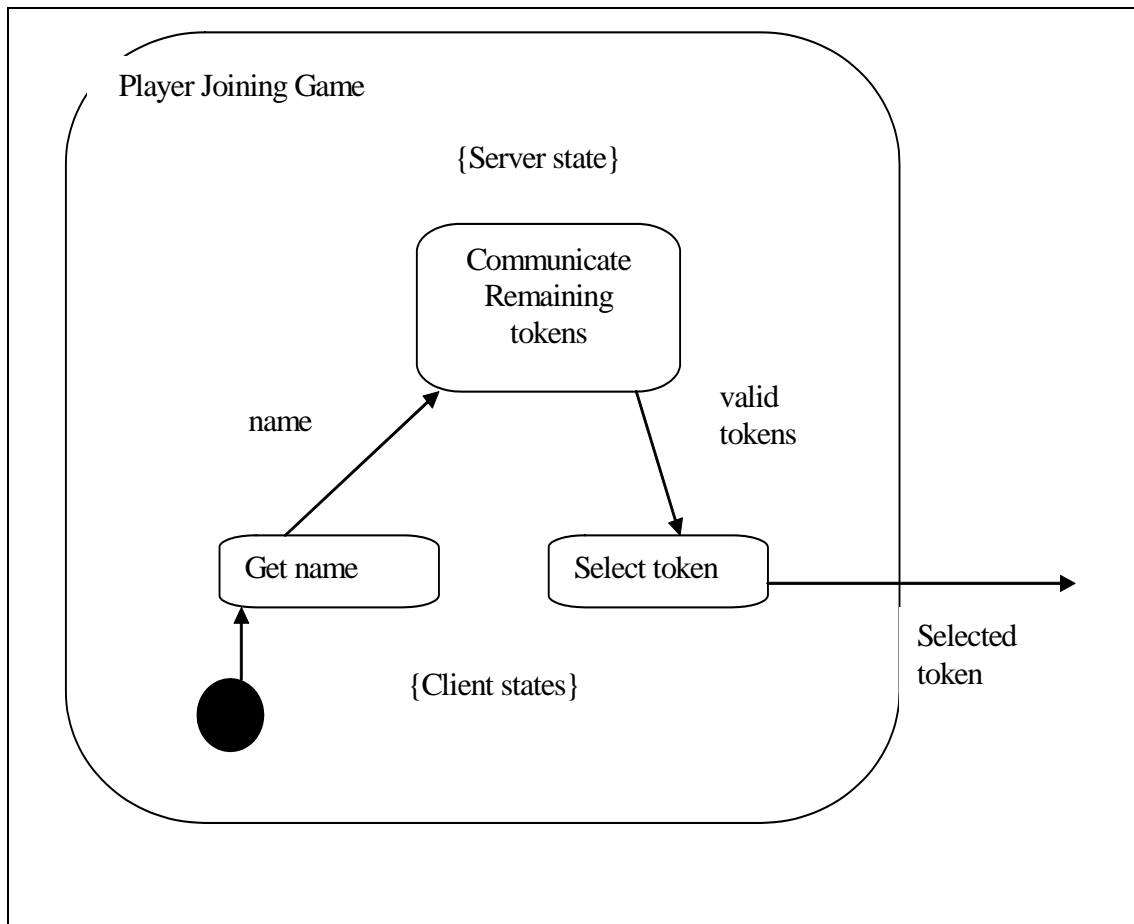


Figure 3.2.1-2: State Machine for Player Joining a Game

Looking at Figures 3.2.1-1 and 3.2.1-2, the *initiate game* state begins at the *player joining game* substate, which in turn begins at the *get name* substate, part of the client. This substate then communicates the player name to the server, and a transition is made to the *Communicate remaining tokens* substate, 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* substate is entered. This substate 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* substate 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.

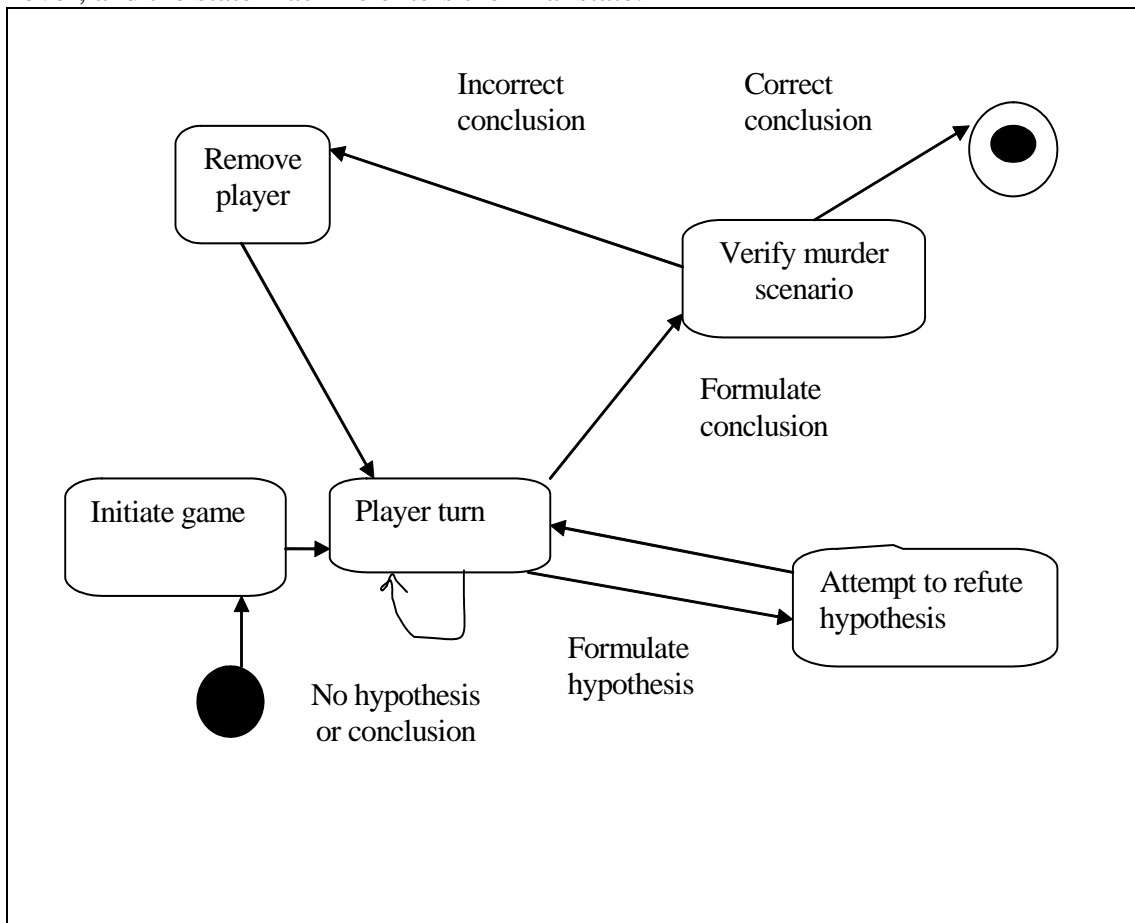


Figure 3.2.1-3: High-Level State Machine Representing Galaxy Sleuth

Figure 3.2.1-3 consists of very abstract states, each of which may be broken down into its respective substates as has been done with the *initiate game* substate. Some of the substates shown in Figure 3.2.1-1 may themselves be deemed to be too abstract, and may be broken down into their substates 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 substates 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 substates, but in the illustration only one of its second level states is decomposed.

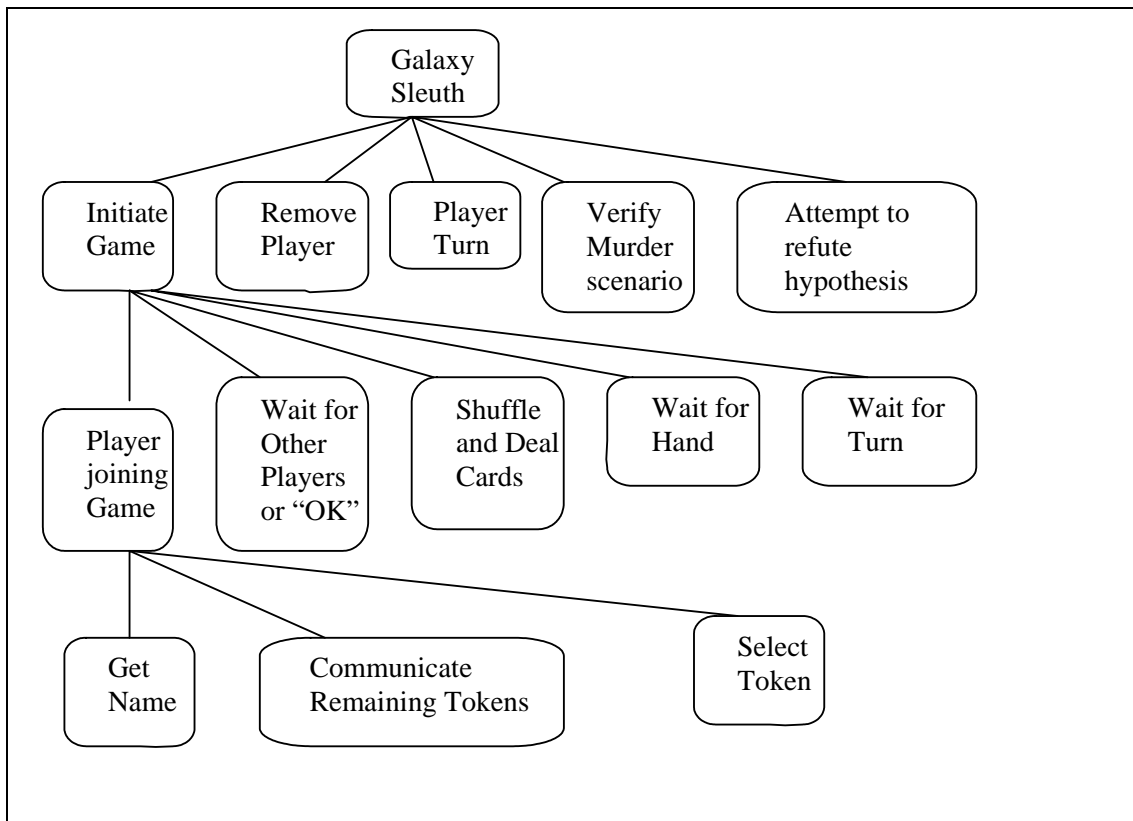


Figure 3.2.1-4: A Hierarchy of State Machines for Galaxy Sleuth [partial]

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. 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”

3.2.3 Designing the Assignment

(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^{iv} 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^v 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^{vi} classes independently and that reasonably distributes the workload.

[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. ...”

3.4 System testing phase

(Section 8.8 of /1/)

As noted in last section (endnote), formally reported-on unit and integration testing processes are **not** required for this assignment. It should be the responsibility of each person who develops code for the team to make sure that code runs correctly.

It is required, however, that system testing based on use cases (and their scenarios!) be planned, designed, carried out and reported on. See lecture notes for the suggested approach.

3.5 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-Wesley2002

ⁱ So that marks may be adjusted proportionately. At the extreme, a student who does not contribute at all to a team’s efforts will get zero for continuous assessment.

ⁱⁱ A project box will be set up in Room L114.

ⁱⁱⁱ It is acceptable that the delivered source code might not be absolutely complete. A note listing any gaps should be included.

^{iv} “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.”

^v “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.”

^{vi} ***N.B.:*** *In fact, because of time limitations, formal unit and integration testing is **not required** for this assignment. **The only testing that must be planned, carried out and reported on is system testing based on use cases.***