Proposal: Individual Design and Development Project
CPE/CSC 305

September 30, 2016

Christopher Siu

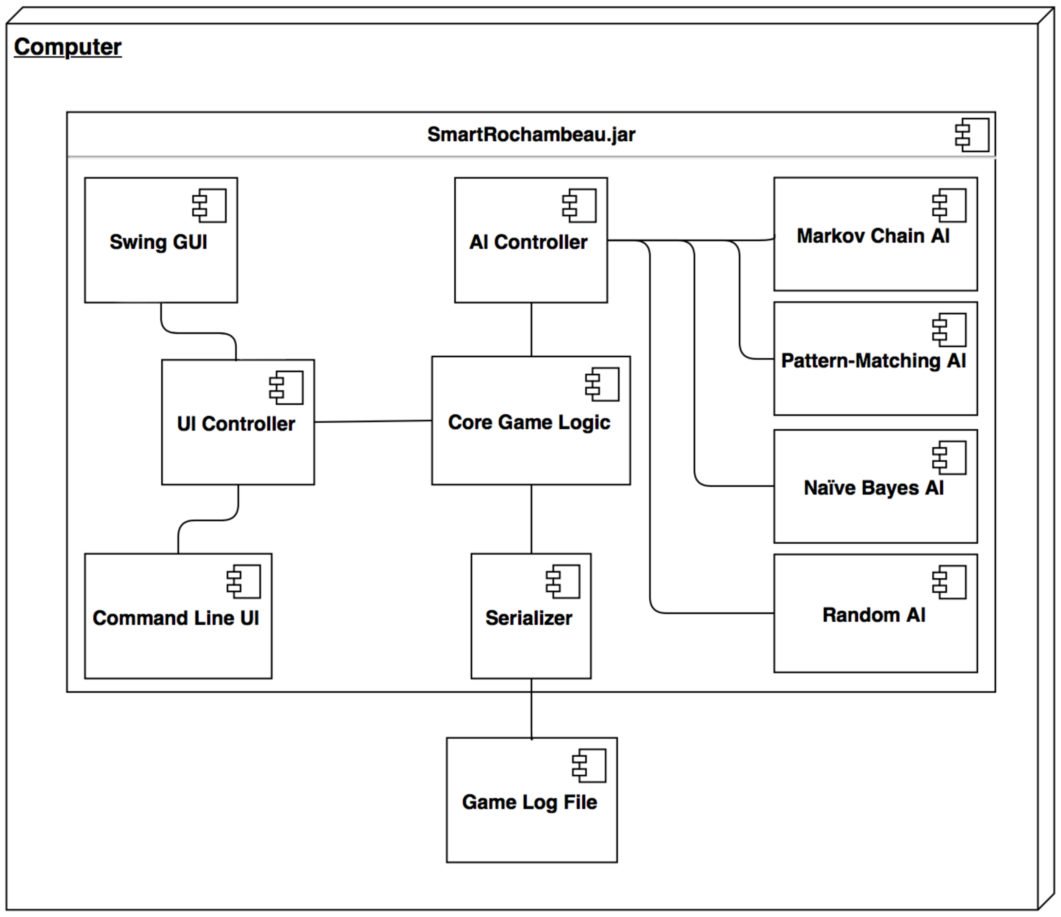
SmartRochambeau

Background and Description:
Over the summer, I wrote a quick proof of concept for a Rock-Paper-Scissors (Rochambeau) AI based on Markov Chains. SmartRochambeau would both develop that concept into a more polished piece of software and allow me to experiment with other algorithms. I must note that I've never taken any classes nor done any work for my jobs that involved machine learning; this is just something I've been interested in on the side.

Features of SmartRochambeau:
- A core module to moderate a game of Rock-Paper-Scissors
- Two user interfaces: a text-based interface and a GUI
- A standard random-number-based AI for the user to play against
- Two more sophisticated AI's, one based on Markov Chains (e.g., if the current state is that the player has thrown 'Rock' and won, what is he historically likely to throw next?) and one based on pattern matching (e.g., if the player has just thrown 'Rock', then 'Paper', then 'Rock', what is he historically likely to throw next?)
- If time allows, one additional AI very loosely based on Naïve Bayes classifiers (e.g., considering all the past throws, what is the probability that the player's next throw will be 'Scissors' now that he's thrown 'Paper'?)
- The ability to log past results, both to support the learning capabilities of the program and to quantitatively evaluate AI performance

Requirements:
- As a casual player, I can launch SmartRochambeau by double-clicking on a JAR and selecting an AI to play against, so that I can then play an indefinite number of rounds of Rock-Paper-Scissors against the computer.
- However, as a user, I cannot switch AI in the middle of a session. I must exit and re-launch the game in order to select a different opponent.
- As a developer, I can run the SmartRochambeau JAR in text-based mode from the command line so that I can easily test the entire system.
- As a developer, I can run SmartRochambeau through either the command line or through the GUI, and both modes will provide statistics so that I can evaluate the performance of each AI.
- As a player, I can see statistics about my own historical wins and losses while playing SmartRochambeau.
- As a player, I can see statistics about my opponents before I pick one to play against.
- As a player, SmartRochambeau does not indicate to me which opponent uses which algorithm; they just have regular (human) names.
- As a developer, I do know which opponent uses which algorithm, as the relationship between opponent name and algorithm never changes.
- As a user, I can copy my game log file to the appropriate folder on a different computer in order to preserve game statistics if I want to run SmartRochambeau on that other machine.

System Architecture:



Technologies:

The GUI will use Swing, and the logging will take advantage of Java's Serializable interface.

Project Breakdown, Timeline, and Milestones:

I estimate that the project will involve between 1250 and 1750 lines of code, to be implemented according to the following timeline:

Date	*Milestone*
October 5	Architecture Diagram
October 10	GUI Prototype Drawings
	Compiled 'Skeleton' Classes with Stubbed Methods
October 14	Game Logic Unit Tests and Implementation
October 19	Random AI and AI Controller Unit Tests, Implementation, Integration
	Text-based UI and UI Controller Implementation, Integration
October 21	Basic Logger Unit Tests, Implementation, Integration
October 26	Basic System Test Cases Written and Passed
October 31	Markov Chain AI and Logger Unit Tests, Implementation, Integration
November 7	Pattern-Matching AI and Logger Unit Tests, Implementation, Integration
November 16	GUI Implementation, Integration
November 23	All System Test Cases Written
November 28	System Test Cases Passed
tbd	Naïve Bayes AI and Logger Unit Tests, Implementation, Integration
tbd	System Test Cases Passed

Risks and Uncertainties:

The obvious uncertainty is that the other students I assign tasks to won't understand the algorithms of my project. However, since they're only supposed to complete roughly 30% of the implementation, I'm confident that there will be enough tasks related to the basic game logic, the UI, or the logging that any student who's passed CPE 102 could complete. Additionally, once I've split the algorithms into smaller, stubbed classes and methods (with pseudocode, if necessary), implementing them should be fairly straightforward. The algorithms I've selected are fairly simplistic, so if I have to, I can explain them in a reasonably short amount of time.