Project Specifications

Tashrique Ahmed, Elyes Laalai

0.1 Introduction

Halal Gamble: The Stock Simulation Game
Halal Gamble is a domain-specific language (DSL) designed to demystify stock market investing through simulation using historical data. Its primary aim is to offer a user-friendly, engaging platform for beginners to learn and experiment with stock investments without financial risks. Unlike complex market simulators, Halal Gamble's intuitive interface and simplified interaction model make it accessible to a broader audience, including those with minimal financial background.

This language simplifies stock market dynamics, focusing on core investment strategies and market fundamentals. By abstracting complex trading concepts, it encourages users to learn through direct interaction with historical market scenarios, promoting a practical understanding of investment principles.

0.2 Design Principles

Halal Gamble is designed with a focus on simplicity, clarity, and user engagement. It features a straightforward syntax and semantics for easy comprehension, alongside efficient data structures and memory management to optimize performance. The language offers flexibility in stock and date selections, ensuring robust type-checking and error-handling for safety. Emphasizing user accessibility, it provides clear documentation and a supportive community to facilitate a positive user experience. Tailored for practical application, our DSL smoothly handles historical stock market data and features an intuitive, natural-language-like syntax suitable for all skill levels. It includes investment-specific functionalities and visualizations, crucial for demystifying complex financial concepts, while allowing for flexible and customizable portfolio management.

0.3 Examples

Sample Program 1

```
dotnet run

/* Open a webpage containing stock market data and news from 2010 to 2015*/
InitialCapital(100)
Buy(GOLD, 25)
Next

/*Open a webpage containing stock market data and news from 2015 to 2016*/
Sell(GOLD, 25)
Buy(SILVER, 70)
Buy(BRONZE, 30)
Next

/*Open a webpage containing stock market data and news from 2016 to 2017*/
Sell(SILVER, 50)
Next
/*Open a webpage containing stock market data and news from 2017 to 2018*/

Sell(SILVER, 50)
Sell(BRONZE, 30)
Next

/*Open a webpage containing stock market data and news from 2018 to 2019*/
```

```
Buy(SILVER, 70)
Next

/*Open a webpage containing stock market data and news from 2019 to 2020*/
Buy(SILVER, 50)
Graph(BarGraph)
Exit
```

Sample Program 2

```
dotnet run

/* Open a webpage containing stock market data and news from 2010 to 2015*/
InitialCapital(100)
Buy(GOLD, 25)
Next

/*Open a webpage containing stock market data and news from 2015 to 2016*/
Sell(GOLD, 25)
Buy(SILVER, 70)
Buy(BRONZE, 30)
Next

/*Open a webpage containing stock market data and news from 2016 to 2017*/
Sell(SILVER, 50)
Next

/*Open a webpage containing stock market data and news from 2017 to 2018*/
Sell(SILVER, 50)
Sell(BRONZE, 30)
Next

/*Open a webpage containing stock market data and news from 2018 to 2019*/
Buy(SILVER, 70)
Next

/*Open a webpage containing stock market data and news from 2019 to 2020*/
Buy(SILVER, 50)
Graph(TimeSeries)
Result(PortfolioStatement)
Exit
```

Sample Program 3

```
dotnet run

/* Open a webpage containing stock market data and news from 2008 to 2010 */
InitialCapital(150)
Buy(OIL, 40)
Buy(TECH, 30)
Next

/* Open a webpage containing stock market data and news from 2010 to 2012 */
Sell(OIL, 20)
```

```
Buy(RENEWABLE, 50)
Next

/* Open a webpage containing stock market data and news from 2012 to 2014 */
Sell(TECH, 30)
Buy(HEALTHCARE, 40)
Next

/* Open a webpage containing stock market data and news from 2014 to 2016 */
Sell(RENEWABLE, 50)
Buy(INFRASTRUCTURE, 60)
Next

/* Open a webpage containing stock market data and news from 2016 to 2018 */
Sell(HEALTHCARE, 40)
Buy(AI_TECH, 70)
Next

/* Open a webpage containing stock market data and news from 2018 to 2020 */
Sell(INFRASTRUCTURE, 60)
Buy(CYBER_SECURITY, 80)
Graph(TimeSeries)
Result(PortfolioStatement)
Exit
```

0.4 Language Concepts

The core concepts in this domain-specific language (DSL) are centered around stock market investment operations. Users need to understand basic investment terms and actions, as well as how to represent these in the DSL's syntax.

Primitives:

- **Stock Names:** These are primitives in the DSL, representing distinct investment assets like GOLD or SILVER.
- **Numerical Values:** These include figures like initial capital and investment amounts. Action Keywords: Words like 'Buy' and 'Sell' are primitives that represent fundamental operations in the language.

Combining Forms:

- **Sequential Commands:** The combination of various commands (like buying and selling stocks, setting initial capital) in a sequence to form a coherent investment strategy.
- **Investment Operations:** Constructs that combine primitives like stock names and numerical values to define a particular operation, e.g., Buy(GOLD, 50).
- **Control Structures:** Commands like Next or Exit, which control the flow of the simulation, acting like control flow statements in conventional programming languages.
- **Data Visualization Commands:** These combine various data points (such as percentages of investment in different assets) into a cohesive graphical representation. They are similar to functions that take multiple inputs and produce a composite output.
- **Portfolio Statements:** Combining different elements of the portfolio (like different stocks and their respective quantities) to generate a comprehensive statement or analysis.

0.5 Syntax

Detailed Syntax of Halal Gamble:

- **Initial Capital:** Defined using ‘InitialCapital(amount)‘. This command sets the starting budget for the simulation.
  - Example: ‘InitialCapital(100)‘ allocates a budget of 100 dollars.
- **Stock Transactions:** Executed with ‘Buy(stock, amount)‘ or ‘Sell(stock, amount)‘, allowing users to purchase or sell stocks.
  - Example: ‘Sell(GOLD, 20)‘ indicates selling 20 dollars of GOLD stock.
- **Control Commands:** ‘Next‘ advances the simulation to the next timeframe, while ‘Exit‘ terminates the simulation.
- **Data Visualization:** Utilize ‘Graph(Type)‘ to display investment data graphically. Supported types include BarGraph, PieChart, and TimeSeries.
  - Example: ‘Graph(TimeSeries)‘ creates a time series plot of the investment data.
- **Result Reporting:** ‘Result(Type)‘ provides a summary or detailed report of the investment performance. Types can include PortfolioStatement, TransactionLog, etc.
  - Example: ‘Result(PortfolioStatement)‘ generates a detailed statement of the portfolio.

0.6 Semantics

Primitive Kinds of Values:

- **Integers:** These are used for specifying quantities like the number of stocks to buy or sell, and for defining initial capital.
- **Strings:** Used for stock names (e.g., GOLD, SILVER) and for potential date inputs.
- **Graphs:** To present visual data, rendered using python matplotlib

Actions and Compositional Elements:

- **Buy/Sell Operations:** These actions combine stock names (String) and quantities (Integer) to execute transactions.
- **Next/Exit Controls:** ’Next’ progresses the simulation, while ’Exit’ terminates it. These control the flow of the program.
- **Visualization Commands:** Commands like Graph(BarGraph) combine historical data to generate visual representations.

The AST (Abstract Syntax Tree) for this DSL will have several key components represented as algebraic data types:

- **TransactionNode:** Represents stock operations, with types like BuyNode and SellNode, each holding stock name (String) and quantity (Integer).
- **CapitalNode:** Represents initial capital setting, holding the capital amount (Integer).
- **ControlNode:** Represents control commands like Next and Exit.
- **VisualizationNode:** For graphical commands, with types like BarGraphNode and TimeSeriesNode.

The AST elements fit together as follows:
The *root* node represents the entire program. Child nodes of the root are operation nodes, capital nodes, control nodes, and visualization nodes, reflecting the sequence of commands in the program. For example, the AST for Sample Program 1 might look like:

```
Root
|-- InitialCapitalNode(100)
|-- TransactionNode
|   |-- Buy
|   |   |-- Stock: GOLD
|   |   |-- Quantity: 25
|-- ControlNode(Next)
|-- TransactionNode
|   |-- Sell
|   |   |-- Stock: GOLD
|   |   |-- Quantity: 25
|   ...
|-- VisualizationNode
|   |-- Type: BarGraph
|-- ControlNode(Exit)
```

Program Evaluation (Input) Programs in this DSL read textual input commands, which include stock names (like GOLD, SILVER), numerical values (such as quantities for buying or selling, initial capital), and control instructions (like Next, Exit). Each line of input corresponds to an operation or control command that affects the simulation of the stock market investment.

Program Evaluaiton (Output) The output of evaluating a program in this DSL includes:
A comprehensive summary of the investment portfolio's performance, detailing gains or losses. Graphical representations of the investment portfolio, showing the distribution and performance of different stocks over time.

Post-Order Traversal for Evaluation: In the post-order traversal of an AST:
Each node represents an operation or control command. The traversal order ensures that each operation's effect on the portfolio is calculated before moving on to subsequent operations.

0.7 Remaining Work

Primitive Kinds of Values: Lorem ipsuem desciprtion