**Declaration template for Senior Project and Diploma Thesis**

**Title**: Text-based adventure game

**Author**: Valeri Genev

**Abstract:** I would like to create a text-based adventure game which utilizes the Djikstra Algorithm and creates a node that records and showcases the shortest path between two set points. The algorithm keeps track of the player’s progress and the time spent on the game. The experience offers replay value as the one interacting with it is encouraged to replay in order to complete the game in the most optimal time.

**Technologies used:** Java.

**Declaration of authorship:**

“The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of Vladimir Georgiev. All sources, used in development, are cited in the text and in the Reference section.”

Author:

**Bulgarian Diploma Thesis**

**Text-based adventure game**

**Valeri Genev, ID#200096522**

**Student:**   **, Date:**

*signature*

**Supervisor:**   **, Date:**

*signature*

**Department of Computer Science, AUBG**

**Blagoevgrad, 2022**

Contents

[Introduction: 4](#_Toc132750137)

[Overview of the problem: 5](#_Toc132750138)

[Non-functional requirements: 5](#_Toc132750139)

[Requirement #1: 7](#_Toc132750140)

[Requirement #2: 7](#_Toc132750141)

[Functional requirements 7](#_Toc132750142)

[Design of the software solution: 9](#_Toc132750143)

[Software architecture: 9](#_Toc132750144)

[The Layered structure: 11](#_Toc132750145)

[The MVC architecture: 12](#_Toc132750146)

[Microservices architecture: 14](#_Toc132750147)

[Monolithic structure: 15](#_Toc132750148)

[User Interface (to be written): 16](#_Toc132750149)

[UML diagrams: (Images to be added) 16](#_Toc132750150)

[Databases used: 18](#_Toc132750151)

[Description of diagrams in computer programming: 18](#_Toc132750152)

[How the code works and description of what each class in the program does: The concept for the program (this section needs to be rewritten): 24](#_Toc132750153)

[Story.java: 26](#_Toc132750154)

[TextadventureGameUI.java: 26](#_Toc132750155)

[VisibilityManager.java: 27](#_Toc132750156)

[Player.java: 29](#_Toc132750157)

[TextAdventure.java: 29](#_Toc132750158)

[Weapon.java: 31](#_Toc132750159)

[Monster.java: 32](#_Toc132750160)

[Combat system: 33](#_Toc132750161)

[What is an algorithm: 34](#_Toc132750162)

[Algorithms I considered implementing: 35](#_Toc132750163)

[Reinforcement learning: 35](#_Toc132750164)

[Breadth-first search: 37](#_Toc132750165)

[Algorithms used for this project: 38](#_Toc132750166)

[How I implemented the algorithm into the program: 40](#_Toc132750167)

[Implementation: 40](#_Toc132750168)

[Programming language used: 40](#_Toc132750169)

[External frameworks and libraries used: 44](#_Toc132750170)

[Swing: 44](#_Toc132750171)

[Installation requirements: 46](#_Toc132750172)

[Operating system: 47](#_Toc132750173)

[Regarding other systems (Mac OS and Linux): 47](#_Toc132750174)

[Testing: 47](#_Toc132750175)

[Unit testing: 49](#_Toc132750176)

[Integration testing: 50](#_Toc132750177)

[Acceptance testing: 51](#_Toc132750178)

[Test-driven development: 53](#_Toc132750179)

[Chosen testing methods: 54](#_Toc132750180)

[Performed unit tests: 55](#_Toc132750181)

[Results and conclusions 55](#_Toc132750182)

[Conclusions drawn and lessons learned from papers examined: 55](#_Toc132750183)

[Lessons learned: 59](#_Toc132750184)

[References: 60](#_Toc132750185)

# Introduction:

When considering what to do for my senior project, I wanted to create a project that would utilize all the theoretical knowledge that I learned while taking the COS 470 course with Professor Stoilov. Given that I was provided with the hypothetical basis for how everything worked, I figured I knew how to achieve whatever goal I had in mind. Given that I wanted to create a project that would be different from what we would normally make in class and provided with the fact that the most common usages for an AI algorithm would be a game, I decided to create one. I dabbled in creating a program that would make use of NLP and create a simplistic version of a program that would check for plagiarism, but decided against it given that the program is unreliable. I am intent on getting acquainted with and being comfortable with AI technologies. Learning how to make and implement algorithms is also a must, provided that I only have some theoretical preparation on the subject and barely any real experience in putting them to use in a real project. It is my belief that artificial intelligence is the future of computer programming and hence find this project a valuable learning experience. In my view, the sooner I am made aware of the intricacies of this technology, the better I would be able to adjust to modern world trends and innovative technologies in the computer science field.

## Overview of the problem:

The problem I was hoping to solve was one that has been boggling me ever since I came to university. Even though I was given an extensive amount of information on how things ought to be done, I never seemed to have a comfortable grip on the practical side of things. I wanted to make a project that I could probably build on and improve in the future. It is my desire to start off with something easy that can be improved as time goes on. My intent was to take the knowledge I considered most important that I learned throughout my four years of formal education and implement it in the best way possible.

## Non-functional requirements:

Qualitative requirements that describe how a software system or application should behave rather than what it should do are referred to as non-functional requirements. The system's performance, scalability, dependability, usability, security, and maintainability are all outlined in these requirements.

There are many different types of non-functional requirements, such as:

Performance: This includes speed, throughput, and response time requirements for the system. A system might have to deal with a certain number of transactions per second or respond to user requests in a certain amount of time.

Scalability: This includes requirements regarding the system's capacity to handle an increase in user traffic or workload. A system might be required, for instance, to deal with a sudden increase in traffic without stuttering or crashing. Reliability: This includes requirements related to how well the system can handle errors and failures. For example, a system might be required to have a certain level of uptime or to be able to recover from hardware or software failures.

Usability: This includes requirements regarding how user-friendly the system is to interact with. A system might have to be accessible to people with disabilities or have an intuitive user interface.

Security: This includes requirements regarding the system's ability to guard against data breaches, unauthorized access, and other security threats. For instance, a framework may be expected to utilize encryption or to consent to specific security principles.

Maintainability: This includes requirements regarding the system's ease of upkeep and updating over time. A system might have to use modular code or have clear documentation, for example.

Non-practical prerequisites are regularly harder to quantify than utilitarian necessities, as they frequently include emotional measures like client fulfilment or framework dependability. However, in order to guarantee that the system meets the requirements of its stakeholders and users, it is essential to clearly define and measure these requirements. Throughout the system's lifecycle, non-functional requirements should be tested and validated and considered early in the development process.

## Requirement #1:

It is essential to develop an application that can operate on multiple operating systems, including Windows, Linux, and MacBook. This project's target audience comprises users who use various devices and platforms other than Windows, such as Linux and IOS.

## Requirement #2:

The application to ought to respond almost immediately after the user initiates an action. This requirement is due to the decreasing human attention span observed in recent years. Hence, it is crucial for the application to be fast and promptly respond to the user's actions. Even a minor delay in response time may result in the user feeling annoyed and frustrated.

## Functional requirements

A software system's or application's functional requirements specify what it should do and how it should behave. They talk about the particular characteristics, functions, and behaviors that the system needs to have in order to do what it was designed to do.

There are several categories of functional requirements:

1. Inputs: The data or information that the system must accept from users or other systems in order to function properly is outlined in this category. User login credentials, search queries, or payment information are all examples of inputs.

2. Outputs: The data or information that the system must produce or display to users or other systems in response to inputs or requests is specified in this category. Search results, error messages, and confirmation emails are all examples of outputs.

3. Prerequisites for Processing: The specific functions or operations that the system must carry out on the input data in order to generate output data are listed in this category. Data validation, calculations, and database queries are all examples of processing requirements.

4. Interfaces for Users: The design and layout of user interfaces like menus, forms, or buttons fall under this category and define how users interact with the system. Web-based GUIs, mobile app interfaces, and voice-activated systems are all examples of user interfaces.

5. Interfaces to the system: The system's communication and interaction with other systems or services, as well as any protocols or data formats required for integration, are outlined in this category. APIs, messaging protocols, and database connectors are all examples of system interfaces.

6. Management of data: Any necessary data structures, databases, or file formats are included in this category to specify how the system stores, manages, and retrieves data. Backups, data security, or data privacy are all examples of data management requirements.

Based on the requirements and needs of users and stakeholders, functional requirements should be clearly defined and prioritized. To make sure the system works as expected, they should be tested and validated in a variety of ways, including user testing, unit testing, and acceptance testing.

# Design of the software solution:

## Software architecture:

The evaluation of a system or application to see if it satisfies its intended requirements and performs as expected is known as testing. It is a crucial part of software development and is usually done at different points along the development lifecycle, from the initial design phase to the system's delivery and upkeep.

The fundamental structural design of a software system that defines the system's components, relationships, and interactions is known as software architecture. It is the blueprint that lays out the ways in which the various modules of a software application will interact with one another in order to satisfy the application's functional and non-functional requirements.

The selection of programming languages, databases, frameworks, libraries, and deployment infrastructure are all components of a software architecture. It gives stakeholders and software developers a high-level view of the software system's behaviour and lets them know how it will work.

Starting with gathering the requirements for the software and evaluating the requirements for the system, the process of designing a software architecture involves several steps. The software architect will be able to identify the system's non-functional requirements, such as performance, reliability, and scalability, with the assistance of this analysis. Functional requirements include things like features and capabilities.

The next step is to identify the system's components, which form the software architecture's foundation. The system can be made up of these components, which can be modules, services, or libraries. The software architect then specifies how these components interact with one another, how they share data and resources, and how they communicate with one another.

The software architect decides how the components will be hosted and deployed after identifying their interactions. This includes picking out the server infrastructure, database management system, and other parts that are needed to help the software system work.

Last but not least, the software architect records the software architecture in a collection of diagrams, models, and other artifacts that present the software system's design in a clear and concise manner. Developers, stakeholders, and other members of the software development team use these documents to comprehend the system's structure and make educated decisions regarding its development and upkeep.

As the foundation for the software system's design, development, testing, and upkeep, software architecture plays a crucial role in software development. To ensure that the software system meets its functional and non-functional requirements and is reliable, scalable, and easy to maintain, a well-designed software architecture is essential.

## The Layered structure:

In software development, layered structures are widely recognized as the most common and popular architectural framework. These structures are often referred to as n-tier architecture, as they consist of multiple horizontal layers that work together to form a single software unit. This design pattern provides a clear and logical separation of components or code, making it easier to maintain and modify software applications.

In layered structures, related components or code are typically placed on the same layer to achieve better organization and functionality. Each layer has its own distinct purpose and contributes to a specific aspect of the larger system. For example, the presentation layer is responsible for handling the user interface and all visual actions, while the business layer stores the application's business logic. The persistence layer connects the business layer and the database layer, while the database layer stores and presents information.

There are several benefits to using a layered architecture in software development. It enables developers to organize code more efficiently and makes it easier to test and maintain software applications. It also provides a clear separation of concerns, allowing developers to work on specific layers without affecting other layers. Furthermore, the layered architecture can be easily scaled and modified to meet the changing needs of a software application.

One of the most common examples of layered architecture consists of four layers, namely the presentation layer, business layer, persistence layer, and database layer. The presentation layer is responsible for the visual aspect of the application, while the business layer implements the application's logic. The persistence layer connects the business layer to the database layer, and the database layer is responsible for storing and presenting information.

Overall, the layered architecture is a widely used and efficient design pattern in software development. It allows developers to create complex and scalable software applications with ease, while also ensuring that the applications are organized and maintainable.

## The MVC architecture:

The entire application architecture is governed by the Model-View-Controller (MVC) architecture, which is a highly influential pattern. Although it is often referred to as a design pattern, it is actually an architecture pattern that affects the entire application's architecture and addresses architectural issues. There are three main parts to the MVC architecture: the Model, View, and Regulator, each with its own arrangement of obligations. The state of the data and the program or business logic that manipulates it are stored by the Model. The application's user interface is the View, which is in charge of displaying the data to the user. The Controller selects a View to render and display the user interface, handles end-user interactions, manipulates the Model, and manages the other two components. Web-based applications benefit from increased robustness and scalability thanks to the MVC architecture, which is frequently utilized to eliminate redundant code and construct a solid structure.

Diagram

Description automatically generated

## Microservices architecture:

The microservices engineering is a structural methodology that separates an application into an assortment of free, little, and secluded administrations that speak with one another through APIs. Each microservice can be developed, deployed, and scaled independently of other services because it is accountable for a specific business capability.

Because small, cross-functional teams can work on specific microservices without affecting the application as a whole, this method makes the development process more adaptable and agile. Since each assistance is independent, it are simpler and more productive to test and support. Additionally, the application's overall stability and scalability are enhanced by the loosely coupled nature of microservices, which reduces the impact of changes made to one service on the system as a whole.

The microservices architecture's ability to allow a company's technology stack to change over time is another significant advantage. New tools and technologies can be integrated into individual microservices using this strategy without having to rewrite the application as a whole. This makes it simpler for businesses to keep their applications current and competitive.

Generally, the microservices design gives a more adaptable, versatile, and viable way to deal with application improvement, which can assist associations with accomplishing their objectives quicker and all the more effectively.

## Monolithic structure:

The monolithic structure is a traditional design model used in software development. It is called "monolithic" because the entire software program is built as a single, self-contained unit. The components of the application are interrelated and interdependent, meaning that any changes made to one component can potentially affect other components.

In a monolithic architecture, all the application's functionality is contained in a single codebase, and all the application components, such as the user interface, the business logic, and the database access layer, are tightly coupled together. This makes the codebase large, complex, and difficult to manage, and also makes it challenging to scale and maintain the application over time.

One of the biggest disadvantages of a monolithic architecture is that it requires the entire application to be recompiled and redeployed whenever a change is made to any component of the application. This can be a time-consuming and error-prone process, especially for large, complex applications.

In contrast to a monolithic architecture, a modular architecture breaks the application down into smaller, more manageable pieces or modules. Each module has its own set of functionalities and is responsible for a specific aspect of the application. Modules can be updated or replaced without affecting other parts of the application, making it easier to maintain and scale the application over time.

Overall, while the monolithic architecture was the dominant model for many years, the move towards modular architectures, such as microservices, has become more popular in recent years due to the benefits of scalability, maintainability, and flexibility that they offer.

## User Interface (to be written):

## UML diagrams: (Images to be added)

A set of standard visual models known as UML diagrams are used to describe and design software systems. They provide stakeholders, such as software developers, business analysts, project managers, and clients, with a standard means of communicating the various aspects of a system. Although each UML diagram type has its own purpose, notation, and syntax, they all adhere to the same fundamental principles.

The various graphical elements used to represent the various aspects of a system, such as classes, objects, actors, use cases, components, and relationships, are the primary components of a UML diagram. Lines, arrows, and other symbols connect these graphic elements, illustrating their connections and interactions.

Each kind of UML diagram is described in greater detail below:

1. Diagram of class: A structural diagram of the system's classes, interfaces, attributes, methods, and relationships between them is known as a class diagram. It is used to describe a system's components and their relationships, as well as its static structure. Classes, interfaces, associations, dependencies, generalizations, and realizations are the primary graphical elements of a class diagram.

2. Diagram of a use case: A utilization case outline is a sort of conduct chart that shows the different use cases, entertainers, and connections between them in a framework. It is used to describe a system's functionality from the user's perspective and the various actors' interactions with the system in order to accomplish their objectives. A use case diagram's main graphical elements are actors, use cases, associations, and generalizations.

3. Diagram of sequence: A type of behavioral diagram called a sequence diagram depicts the system's objects' interactions over time. It is used to describe a system's dynamic behavior and the way objects work together to complete a task. Objects, lifelines, messages, activations, and return messages are the most prominent graphical components of a sequence diagram.

4. Diagram of activity: An activity diagram is a type of behavioural diagram that shows the various actions and activities involved in a system's workflow. It is used to describe the steps that need to be taken to complete a particular task and the various routes that can be taken depending on certain conditions. An activity diagram's activities, actions, decisions, forks, joins, and swim lanes are its primary graphical components.

5. Diagram of state: A type of behavioural diagram known as a state diagram depicts an object's or system's various states and transitions throughout time to describe its behaviour. It is used to describe the various modes or phases that an object or system can go through, as well as the conditions that cause a state to change. A state diagram's main graphical elements are states, transitions, events, and actions.

6. Diagram of components: The various system components and their relationships are depicted in a component diagram, a type of structural diagram. It is used to talk about a system's physical architecture and how different parts work together to do something. A component diagram's main graphical elements are realizations, dependencies, interfaces, ports, and components.

7. Diagram of deployment: A deployment diagram is a type of structural diagram that depicts the hardware components of a system and their connections to show the system's physical architecture. It is utilized to depict the dissemination of programming parts on various equipment hubs, as well as the correspondence channels between them. Nodes, components, associations, and dependencies are the primary graphical components that make up a deployment diagram.

In conclusion, software developers, business analysts, project managers, and clients use UML diagrams to communicate the various aspects of a system. They provide a standardized method for describing and designing software systems. The various types of UML diagrams are utilized to describe the various aspects of a system and each have their own purpose and notation.

The various types of UML diagrams are utilized to describe the various aspects of a system, including its structure, behavior, functionality, and physical architecture. Each type of UML diagram has its own purpose and notation. Stakeholders can improve their comprehension of and communication of the various aspects of a system by employing UML diagrams, identify potential issues or enhancements, and ensure that all project participants have a common understanding of the system's requirements and design.

## Databases used:

No databases were used in the creation of this project. I originally considered using SQL lite for some operations, but ultimately decided against it.

## Description of diagrams in computer programming:

A graph is generally addressed as an assortment of hubs and edges. Each edge represents a connection between two nodes, and each node represents an object. Depending on whether the connections between nodes are unidirectional or bidirectional, edges can be directed or undirected. Weighted edges can also be used to show how strong or far apart nodes' connections are.

In computer programming, graphs are typically represented in one of two ways:

Matrix of Neighborhoods:

The chart is represented by this array, which has two dimensions. Each line and segment of the lattice addresses a hub of the graph. The matrix's row and column values (or edge weights) are 1 if there is an edge between two nodes. If there are no edges, the value is 0.

List of neighbors:

A collection of linked lists that represent charts is here. There is a linked list for each node that lists all the other nodes it connects to. If they are weighted, edge weights can also be stored in the linked list.

Different algorithms, such as depth-first search (DFS) and breadth-first search (BFS), can be used to traverse the graph. BFS layers all nodes at a given depth before moving on to the next depth, whereas DFS explores the graph as much as possible along each branch before returning.

Additionally, a number of issues can be resolved using graphs, such as: B. Determine the graph's connectivity, cycle locations, and the shortest route between two nodes.

In conclusion, complex system modelling and analysis are made possible by the powerful data structures that graphs provide in computer programming. Adjacency matrices or adjacency lists can be used to represent them, and algorithms like DFS and BFS can be used to traverse them. Graphs are essential tools in computer science because they can be used to solve a wide range of issues.

Diagram

Description automatically generated

Shape, arrow

Description automatically generated

## How the code works and description of what each class in the program does: The concept for the program (this section needs to be rewritten):

The kind of program I've decided to make is a text-based adventure game in which the player uses text commands to move through a virtual world. Textual descriptions of the game world and the player's actions are used to convey the game's non-graphical presentation.

The game world is typically depicted on a map that is broken up into a series of locations. Every area is portrayed exhaustively, with data about the climate, items, and characters that can be tracked down there. By entering commands like "go north" or "go to the forest," the player can change locations.

Most of the time, the game starts with a brief introduction that sets the scene and gives the player some background information about their character and what they want to do. The player is then given a set of objectives or tasks to complete to advance through the game. Finding a specific object, solving a puzzle, or defeating an adversary are all examples of these tasks.

The player uses natural language commands, such as "look at the book" or "open the chest," to move around the game world. The game's parser interprets these commands and determines the best course of action based on the player's input. The player's action is then explained by the game, such as "You open the chest and find a key" or "You examine the book and find a clue hidden inside."

Doors, chests, and weapons are examples of objects that the player can interact with in the game world. These things can be used to get around problems or solve puzzles. The player may, for instance, be required to locate a key to unlock a door or use a weapon to defeat an adversary.

Non-player characters (NPCs) that can interact with the player are present in the game. These NPCs can give you information, lead quests, or fight. By typing in natural language, the player can converse with NPCs, and the game's parser will interpret the conversation and respond appropriately.

Turn-based combat is typically used in text-based adventure games, with the player and the enemy taking turns attacking one another. The player has the option of attacking with a weapon or a unique ability. A random number generator takes into account the player's strength and agility to determine the fight's outcome.

A system of skills or attributes that can be improved over time may also be included in the game. The player might acquire experience focuses by finishing journeys or overcoming adversaries, which can be utilized to step up and work on their capacities. Additionally, the player may be able to acquire items that enhance their abilities, such as a potion that boosts strength or a cloak that boosts stealth.

Through the utilization of natural language commands and descriptive text, a text-based adventure game offers an overall rich and immersive experience. The game's non-linear nature allows for a high degree of replayability and requires the player to use their imagination to visualize the world. The skill of the game designer and the technology used to create the same can have a significant impact on the game's complexity and depth.

### Story.java:

SelectPosition takes a single argument nextPosition, which is a string. It uses a switch statement to determine which method to call based on the value of nextPosition. Each case in the switch statement corresponds to a different method in the code, with the method name matching the case label. For example, if nextPosition is "wakeUpCallRecollection", the wakeUpCallRecollection method will be called.

wakeUpHut sets the text and choices of a text-based user interface (UI) for a game. The UI displays a message about the player waking up alone in a room with women's clothes on the opposite bed. It provides four choices for the player to make, each with a corresponding nextPosition value that will be passed to selectPosition if the choice is selected. The nextPosition values correspond to different parts of the game and determine what will happen next in the game's story.

### TextadventureGameUI.java:

This class has several member variables, including window, cont, and a number of panels (the titleNamePanel, startButtonPanel, mainTextPanel, choiceButtonPanel, and playerPanel, among others). Additionally, a number of labels, buttons, and text areas have been defined for the game's display and interaction.

The class's main method, CreateUI, is responsible for establishing the user interface. It sets the size of a JFrame object and the default close operation to JFrame. EXIT\_ON\_CLOSE. In addition, it makes the window visible, disables the default layout, and makes the content pane's background color black.

After that, it positions and sizes the mainTextPanel, startButtonPanel, and titleNamePanel panels. The titleNamePanel now has a new label called titleNameLabel. The startButtonPanel receives the startButton as well as the exitButton.

The mainTextPanel now contains the mainTextArea. The content pane now has a choiceButtonPanel created. The choiceButtonPanel receives the creation of the choice1, choice2, choice3, and choice4 buttons.

In the end, the CreateUI method receives the ChoiceHandler object. The game's ChoiceHandler object is in charge of handling the user's choices. The ChoiceHandler receives an action command from the user when they click a button, which causes it to carry out the game's appropriate action.

In general, this code creates a basic user interface for a text-based adventure game, complete with buttons that let the player make choices and advance through the game.

### VisibilityManager.java:

Defines the "VisibilityManager" public class, which is in charge of controlling the visibility of various panels in a text adventure game. "UI," the only instance variable in the class, is of type "TextAdventurePrototypeUI."

A single parameter is passed to the constructor of the VisibilityManager class: an instance of the TextAdventurePrototypeUI class. The instance variable "UI" is initialized with this parameter.

There are two public methods in the class: "showHutTitle()" and "showTitleScreen()" The game's title screen is shown using the "showTitleScreen()" method, and the game's main content is shown using the "showHutTitle()" method.

The visibility of the titleNamePanel and the startButtonPanel is set to true in the "showTitleScreen()" method, whereas the visibility of the mainTextPanel, choiceButtonPanel, and playerPanel is set to false. This conceals the primary substance of the game and shows the title screen.

The titleNamePanel and the startButtonPanel are hidden in the "showHutTitle()" method, while the mainTextPanel, the choiceButtonPanel, and the playerPanel are visible. The text that describes the player's surroundings, the buttons that the player can use to make decisions, and the panel that shows the player's status are all displayed here.

In general, the game's panels can be controlled with the help of the VisibilityManager class. The game can display either the title screen or the main content by calling the appropriate method on an instance of the class.

### Player.java:

Is the code that creates the Player class in the textadventureprototype package. "hp”, “currentWeapon," "strength", "charisma", "intelligence", and “dexterity” are the instance variables in the class.

The player's health points are represented by the "hp" variable, which is an integer. An object of the "Weapon" class that comes from an external package called "AdditionalPackages" is used to create the "currentWeapon" variable. The player's in-game currency is represented by the integer "goldenCoins" variable. The "currentHeadwear", "currentChestwear", and "currentFootwear" factors are objects of the "Headwear", "Chestwear", and "Footwear" classes separately, which are not characterized in this code bit. Although it is mentioned in the comments that armor has not yet been implemented, the "hpAdded" variable is an integer that represents the additional health points added by armor.

As a whole, this code creates a template for a player object that can store information about the player's health, weapons, stats. This object could be used to track the player's progress and status throughout a text-based adventure game.

### TextAdventure.java:

To begin, the code imports the java.awt.event package's ActionEvent and ActionListener classes, which are necessary for event handling.

Several other classes are first instantiated by the TextAdventurePrototype class: Story, TextAdventurePrototypeUI, ChoiceHandler, and VisibilityManager ChoiceHandler is a nested class that takes care of button clicks in the user interface and implements the ActionListener interface. The class TextAdventurePrototypeUI is responsible for developing the game's graphical user interface (GUI). The class VisibilityManager is responsible for controlling which GUI panels are visible to which users. The story elements of the game are contained in the Story cass, which is also in charge of updating the game based on the player's choices.

After that, the code specifies four String variables: nextPosition1, nextPosition2, and nextPosition3. The next positions the player can move to base on their choices will be stored using these variables.

The main() method initializes the game by creating a brand-new instance of TextAdventurePrototype.

The TextAdventurePrototype constructor begins by passing in the ChoiceHandler object and creating a brand-new TextAdventurePrototypeUI instance. It then passes this, vm, and UI into a new Story instance. This is talking about the TextAdventurePrototype object, which is the TextAdventurePrototype class's current instance. VisibilityManager is an instance of vm, and TextAdventurePrototypeUI is an instance of UI. These objects are saved as instance variables by the Story constructor.

On the Story object, the defaultSetup() method is called, which sets the player's starting position and initializes the story.

On the VisibilityManager object, the showTitleScreen() method is called to display the title screen and hide the game's panels.

The ChoiceHandler class uses the ActionListener interface, which calls for the actionPerformed() method to be implemented. A parameter to this method is an ActionEvent object, which is used to identify which button was clicked.

The actionPerformed() technique utilizes a change explanation to figure out which button was clicked and plays out the proper activity. The program terminates upon clicking the "Exit" button. The game's panels are displayed by calling the VisibilityManager object's showHutTitle() method when the "Start" button is pressed. The Story object is updated using the wakeUpHut() method, which also displays the player's options. The selectPosition() method on the Story object with the appropriate nextPosition variable is called when one of the choice buttons (c1, c2, c3, or c4) is clicked. This displays the player's options for the next move and updates the story.

### Weapon.java:

This class represents a specific kind of object that can be used in games or other similar contexts.

There are two instance variables in the class: “damage" and "name," both of which have been made public. damage" is a number that indicates how much damage the weapon can deal to an opponent or enemy in the game. name" is a String variable that indicates the weapon's name.

In a nutshell, the Weapon class is a fundamental representation of a weapon object that can be utilized in video games and other applications that are comparable. It provides two public variables that other classes can directly access and modify to represent the weapon's name and damage.

### Monster.java:

Represents a specific kind of object in a video game or other applications that are similar to it. There are four instance variables in the class: name, hp, attack, and attackMessage, respectively. Since each of these variables has been declared to be public, other classes can directly access and modify them. However, due to its violation of encapsulation principles, this may not be the best method.

The monster's name can be represented by the String variable "name." The monster's number of health points is represented by the integer variable "hp." The monster's "attack" is an integer variable that indicates the amount of damage it can deal to a player or another monster. The message that is displayed when the monster attacks is represented by the String variable "attackMessage."

The only way to create a Monster object is to create an instance of the class and assign values to its public variables because this class does not have any defined constructors or methods.

In conclusion, the Monster class provides a fundamental representation of a monster object that can be utilized in video games and other applications that are comparable. It provides four public variables that other classes can directly access and modify to represent the monster's name, health points, attack power, and attack message.

### Combat system:

I have decided to make the combat turn based. The system involves two entities: a player and a monster.

The combat system consists of two methods: playerAttack() and monsterAttack(). The methods are called alternately until one of the entities has its HP reduced to 0 or below. When that happens, the game proceeds to the next position (win or lose).

In the playerAttack() method, the player inflicts damage on the monster using their current weapon. The damage is calculated randomly based on the weapon's damage stats. The actual damage inflicted ranges from 0 to the maximum damage specified for the weapon. The UI is updated to show the attack message and the damage inflicted on the monster. The monster's HP is then reduced by the damage inflicted.

After the player has attacked, the game checks if the monster's HP is still above 0. If it is, the game proceeds to the monster's attack using the monsterAttack() method. If not, the game proceeds to the win position.

In the monsterAttack() method, the monster inflicts damage on the player using its attack stat. The damage inflicted is calculated randomly based on the attack stat. The UI is updated to show the attack message and the damage inflicted on the player. The player's HP is then reduced by the damage inflicted.

After the monster has attacked, the game checks if the player's HP is still above 0. If it is, the game proceeds to the player's attack using the playerAttack() method. If not, the game proceeds to the lose position.

The combat system continues until one of the entities has its HP reduced to 0 or below, at which point the game proceeds to the corresponding win or lose position. The combat system provides an element of chance through the use of random damage calculation, making the outcome of each battle unique.

# What is an algorithm:

A set of instructions that a computer or a person can follow to solve a problem is called an algorithm. It's like a recipe that tells you exactly what to do and how to do it in the right order to get the result you want.

Depending on the issue they are meant to address, algorithms can be straightforward or more involved. They can be used for things like data sorting, finding information, and solving math problems.

We need to define the problem that needs to be solved, break it down into smaller steps, and decide in what order those steps should be done before we can create an algorithm. Algorithmic thinking is the name given to this method.

In the case of human-centric algorithms, you can implement your algorithm manually or in a programming language once you have designed it. Finding the most effective and efficient solution to a problem is the objective of algorithms.

Algorithms include, but are not limited to, sorting algorithms, search algorithms, optimization algorithms, and machine learning algorithms. Each type is made to solve a particular issue as quickly as possible. In synopsis, a calculation is a bunch of directions that guide a PC or human to take care of an issue. Applications ranging from straightforward data manipulation to intricate machine learning make use of these essential tools in computer science.

## Algorithms I considered implementing:

## Reinforcement learning:

Reinforcement learning (RL) is a method of machine learning in which an agent learns to take actions that maximize a cumulative reward signal by interacting with its environment. The agent's objective in RL is to learn an optimal policy that maps states to actions and receive feedback in the form of rewards or punishments.

The following components are included in the RL procedure:

• A formal environment that outlines the issue that the agent must resolve. Agents are rewarded and observed based on their actions by the environment.

• A state space, which is the arrangement of all potential expresses that the climate can take.

• The Action field, which contains a list of every possible action the agent can take.

• A reward function that specifies the reward the agent receives for a specific action in a specific state. policy. This is a mapping of states to actions that agents take in order to get the most out of their total reward.

The following steps are typically included in the RL procedure:

1. The agent keeps an eye on the environment at the moment.

2. Based on the observed status and the current policy, the agent selects an action. 3. The agent is rewarded according to the state of the environment, which changes to the new state.

4. Based on the states it observes, the actions it takes, and the rewards it receives, the agent adjusts its policy.

Five. The first step of this procedure is repeated until the agent finds the best policy.

Finding a balance between exploitation and exploration is one of RL's greatest difficulties. The agent should look around to find the best policy, but it should also use what it knows to get the most out of its reward. To deal with this problem, Q-learning, SARSA, and actor-critical algorithms have been developed.

Robotics, games, and recommendation systems are just a few of the many areas in which RL has been utilized with success. It has the potential to enable agents to learn how to carry out intricate tasks without the need for explicit programming and to gradually adapt to shifting environments.

In general, reinforcement learning is an effective method for resolving challenging issues in which the most effective solution is unknown a priori. It has the potential to make it possible for intelligent systems to make decisions in real time, learn from experience, and adapt to shifting environments. However, RL is also a challenging and computationally demanding field, and developing efficient RL algorithms frequently necessitates a thorough comprehension of both the problem domain and machine learning. Despite this, RL has made significant progress in recent years and may continue to play an important role in future research and development.

## Breadth-first search:

An algorithm known as breadth-first search (BFS) uses a breadth-first approach to traversing a tree or graph. It begins at the root or any hub and investigates every one of the adjoining hubs at the ongoing profundity prior to moving to a higher level.

The BFS algorithm operates in a straightforward manner: First, add a starting node to a queue and select it. The next step is to remove the first node from the queue while the queue is still full, visit it, mark it as visited, and locate all of the removed node's neighboring nodes that have not yet been visited. Add these unvisited adjoining hubs to the furthest limit of the line, and rehash the interaction until the line is unfilled.

By following this calculation, BFS will visit every one of the hubs in a chart or a tree in an expansiveness first request, meaning it will visit every one of the hubs at a similar profundity level prior to moving to a higher level.

Consider a straightforward undirected graph with five vertices (A, B, C, D, and E) and six edges (AB, AC, BD, CD, CE, and DE) to better comprehend the BFS algorithm. Start the BFS at node A, add it to the queue, remove it from the queue, visit it, mark it as visited, and discover its neighbors (B and C) B and C are now in that order in the queue after these nodes are added at the end. We visit B, mark it as visited, and locate its neighbors, A and D, after removing it from the queue. C and D are now in that order in the queue after we move D to the end of it. We visit C, mark it as visited, and locate its neighbors, A and E, after removing it from the queue. D and E are now in that order in the queue after we move E to the end of it. We eliminate D from the line, visit it, mark it as visited, and track down its adjoining hubs (B and C). We don't add these hubs to the line since they have proactively been visited. We visit E, mark it as visited, and locate its neighboring node (C) after removing it from the queue. We don't add C to the line since it has proactively been visited. The BFS algorithm is complete, and the nodes were visited in the order A, B, C, D, and E. In conclusion, BFS is a straightforward and efficient method for traversing trees or graphs in a breadth-first order. It can be applied to a wide range of tasks, including locating the shortest route between two nodes, determining a graph's connectivity, and exploring all of a graph's nodes.

## Algorithms used for this project:

After much consideration, I made the decision to utilize Dijkstra’s algorithm for the completion of this project.

Dijkstra's algorithm is the shortest path algorithm that is used to find the shortest path between two vertices in a weighted graph. It was developed by Dutch computer scientist Edsger Dijkstra in 1956 and is widely used in routing and network packet scheduling. The algorithm works by maintaining a set of vertices whose shortest distance from the source vertex has already been determined, and repeatedly selecting the vertex with the smallest tentative distance as the next vertex to process.

Here's a step-by-step explanation of Dijkstra's algorithm:

1. Initialize: First, we initialize a few data structures needed for the algorithm. We maintain a set S of vertices whose shortest distance from the source vertex s has already been determined, and a priority queue Q containing all the vertices in the graph. We also maintain a map dist that maps each vertex to its tentative distance from the source vertex, and a map prev that maps each vertex to its predecessor in the shortest path from the source vertex.
2. Set the source vertex: We set the tentative distance of the source vertex s to 0 in the dist map and add it to the priority queue Q.
3. Process vertices: While the priority queue Q is not empty, we repeatedly extract the vertex u with the smallest tentative distance from the priority queue.
4. Check if the vertex has been visited: If the vertex u has already been visited (i.e., it is in the set S), we skip it and move on to the next vertex in the priority queue.
5. Update distances: For each neighbor v of u, we calculate a tentative distance alt from the source vertex s by adding the weight of the edge from u to v to the tentative distance of u. If this tentative distance is smaller than the current tentative distance of v, we update the dist map with the new distance and set the predecessor of v to u. We also add v to the priority queue Q if it is not already in the set S.
6. Add the vertex to the set: Finally, after processing all the neighbors of u, we add u to the set S.
7. Termination: The algorithm terminates when the destination vertex t has been added to the set S, or when the priority queue Q is empty (which indicates that there are no more vertices reachable from the source vertex).
8. Extract the shortest path: To extract the shortest path from the source vertex s to the destination vertex t, we start with t and follow the prev map backwards until we reach s.

In summary, Dijkstra's algorithm finds the shortest path between two vertices in a weighted graph by maintaining a set of visited vertices and a priority queue of tentative distances, updating the distances of neighbouring vertices as it processes each vertex, and selecting the vertex with the smallest tentative distance as the next vertex to process.

## How I implemented the algorithm into the program:

# Implementation:

## Programming language used:

Java is the preferred programming language. Java is a programming language that can be used for a wide range of software projects. It is a general-purpose language. It is a high-level, object-oriented language developed with portability, simplicity, and readability in mind. Because it is designed to be simple to learn and use, the language is being used a lot.

Since Java is a statically typed language, each variable must be declared with a particular data type, like an integer, string, or boolean. This aids in the detection of errors at compile time, potentially reducing development time and effort. Additionally, because Java is strongly typed, a variable's type cannot be changed after it is declared.

Java's use of a virtual machine is one of its most important features. Bytecode is a platform-independent code that can run on any machine that has a Java Virtual Machine (JVM) installed. When a Java program is compiled, it is translated into bytecode. The Java Virtual Machine (JVM) is a software layer that acts as a bridge between the Java program and the hardware that runs it. It provides a consistent runtime environment across platforms. Java applications can now be run on a wide range of devices, including servers and smartphones, thanks to this.

The term "object-oriented" refers to the fact that Java is based on the idea of objects. An instance of a class is an object, which is a blueprint that specifies the object's methods and properties. Java gives a rich arrangement of item situated highlights, like legacy, polymorphism, and exemplification, which can be utilized to construct complex programming frameworks.

There are a lot of pre-built classes and methods in the Java standard library that can be used to do common tasks like file I/O, networking, and database access. Additionally, there is a large ecosystem of third-party libraries and frameworks available in Java that can be utilized to enhance the language and create more intricate applications.

An Integrated Development Environment (IDE), which provides a set of tools for writing, testing, and debugging code, is typically used to develop Java programs. Java programs can be run from within an IDE or compiled and run from the command line.

In general, Java is a powerful and adaptable programming language that has been used to create a wide range of software applications, including video games and business applications. It is a popular choice for building complex systems due to its extensive standard library, ecosystem of third-party libraries, and object-oriented features, as well as its use of a virtual machine and object-oriented features.

Logo

Description automatically generated

## External frameworks and libraries used:

I haven’t used any external libraries or frameworks with the exception of the default ones in Java.

## Swing:

Swing is a Java graphical user interface (GUI) widget toolkit that provides a set of lightweight, programmable components that can be used to make rich, responsive desktop applications. The graphical user interface (GUI) components that Swing offers include buttons, labels, text fields, tables, trees, and many more.

The Abstract Window Toolkit (AWT), which provides the fundamental infrastructure for creating Java GUI components, is the foundation upon which Swing is built. Swing, on the other hand, is made to be more powerful and adaptable than AWT. It has more components, better layout managers, and better support for a pluggable look and feel.

Swing adheres to the Model-View-Controller (MVC) architecture, which divides the application data (the Model), the presentation (the View), and the logic that controls user interaction (the Controller) into separate components. This design encourages code reusability and maintainability while also allowing for a clear separation of concerns.

Because Swing has a pluggable look and feel architecture, components' appearances can be altered without altering the underlying code. Swing has a number of pre-built look and feel themes, like Metal, Nimbus, and Windows, and developers can make their own to match their application's branding or style.

Additionally, Swing allows developers to create responsive, interactive applications by responding to user actions like mouse clicks or keystrokes thanks to its support for event-driven programming. Swing provides developers with a comprehensive event model that makes it simple and organized for them to register event listeners and respond to events.

Swing is a powerful and adaptable graphical user interface (GUI) toolkit that offers event-driven programming support, improved layout managers, a pluggable look and feel, and a comprehensive set of components. Swing is an excellent choice for developing desktop Java applications due to these features.

Graphical user interface, application

Description automatically generated

## Installation requirements:

In order to ensure proper installation and operation of the project, I would like to provide some fundamental prerequisites. Firstly, it is essential to have any IDE that can run Java. The one I used for this project was NetBeans. Apart from this, there are no other specific requirements.

## Operating system:

As for the operating system and computing platform, Windows 10 Pro x64 bit was selected for this project since it is the most commonly used operating system among computer users. Unfortunately, I do not have access to a device that runs Linux or Mac OS.

## Regarding other systems (Mac OS and Linux):

Although Windows is the dominant OS, Linux and Mac OS offer several advantages over it. Firstly, they are faster and lighter, with fewer background processes, and also provide enhanced security. Moreover, they are free, while the Windows operating system requires the purchase of an activation key to work. These are some of the most significant differences between these systems.

# Testing:

There are many different kinds of testing, including acceptance testing, system testing, unit testing, and integration testing. Integration testing focuses on testing the interactions between various components, whereas unit testing focuses on testing individual system components or modules. While acceptance testing focuses on testing the system from the user's perspective, system testing tests the system.

Testing can be carried out either manually or automatically, in addition to the various types of testing. While automated testing makes use of software tools to carry out tests and evaluate the results, manual testing is carried out by human testers. Despite the fact that automated testing can be faster and more effective than manual testing, it necessitates a larger initial investment in the creation of the tests.

Typically, test cases, which are sets of instructions or scenarios used to test a particular aspect of the system, are used for testing. A description of the test, the expected result, and the actual result are typically included in a test case.

When a test case is run, one of three outcomes may occur: a pass, a fall flat, or a blunder. A pass indicates that the system is operating as expected and that the test has produced the expected outcome. A failure means that the test resulted in something other than what was expected, pointing to a system flaw or defect. An error indicates that the test was unable to be carried out because of a technical issue, such as an issue with the test environment.

The development team must investigate the issue, determine the root cause of the issue, and implement the necessary system modifications in response to a test failure or error. The test can be repeated to verify that the system is now operating as expected once the issue has been resolved.

In conclusion, testing is an essential part of software development because it ensures that the system performs as expected and meets its intended requirements. It involves running test cases to evaluate the system's behaviour and find bugs or defects. It can be done manually or automatically. Although testing can be time-consuming and difficult, it is necessary to ensure that software meets user requirements and is of high quality.

## Unit testing:

The unit test is an essential tool for any skilled software developer. A type of software testing called a "unit test" is one that is done on individual units or parts of a larger software system to make sure that each one works as expected. It is a type of white box testing in which the code's internal workings are looked at in detail to make sure it meets the design's requirements and gives the right result under a variety of input conditions.

Typically, the process of creating a unit test entail creating a collection of automated test cases that test the unit's functionality. To guarantee that the unit has been thoroughly tested and is robust in its behaviour, these test cases are designed to cover a wide range of scenarios, including edge cases, boundary conditions, and common use cases.

A testing framework executes the test cases during the unit test automatically, capturing the unit's output and comparing it to expected outcomes. Assuming any disparities are found, the test system raises a blunder or affirmation disappointment, demonstrating that the unit has bombed the test and requires further consideration from the engineer.

In general, the unit test is an important part of the software development process because it helps to make sure that code is correct, reliable, and easy to fix over time. Before integrating individual code units into a larger system, developers can catch bugs early and avoid costly errors down the road by testing them thoroughly. Therefore, in order to guarantee the software's quality and dependability, make sure to incorporate unit testing into your development process, regardless of whether you are working on a small hobby project or a large enterprise system.

## Integration testing:

An integration test is a method of software testing that looks at how different parts or modules of a larger system interact with one another and are compatible. It is a kind of testing that makes sure that all of the system's parts work together seamlessly and do what they're supposed to do in a cohesive way. The objective of a combination test is to distinguish any issues or issues that might emerge when the parts are joined, and to guarantee that the framework works as planned.

Before system testing, which tests the entire system, and after unit testing, which tests individual components in isolation, integration tests are typically carried out. Top-down testing, bottom-up testing, or a combination of the two are all methods for conducting integration testing. When testing from the top down, the highest-level components are tested first, then the lower-level components, and finally all of the components are tested together. The lowest-level components are tested first in bottom-up testing, then the higher-level components until all of the components have been tested together.

There are two subcategories of integration testing: testing of vertical integration and horizontal integration. The evaluation of the integration of components across various functional layers, such as the integration of the user interface with the business logic layer, is known as horizontal integration testing. Integration of components within the same functional layer, such as the integration of various database tables, is the subject of vertical integration testing.

During a joining test, different situations and use cases are executed to guarantee that the framework functions true to form. Simulated user interactions, input data, and anticipated outputs may be part of these scenarios. In order to determine whether the system satisfies the desired level of functionality and performance, the outcomes of these tests are contrasted with predetermined criteria.

The size and complexity of the system being tested, as well as the number of components involved, can influence the length and complexity of an integration test. In order to guarantee that each component is properly tested and integrated, it may necessitate extensive planning, preparation, and coordination. Besides, mix testing might require specific apparatuses and assets to robotize the testing system and create significant test reports.

In general, integration testing is a crucial part of the software development process that helps to guarantee the functionality and dependability of complex software systems. It necessitates a comprehensive comprehension of the system's design and architecture, in addition to an in-depth comprehension of each component's interactions with one another. Before the system is deployed, successful integration testing can assist in identifying and resolving issues, resulting in a higher-quality product and a more satisfied user base.

## Acceptance testing:

Acceptance testing is a type of software testing that evaluates whether a system meets the specified requirements and is acceptable for delivery to the end-user or customer. This type of testing is also known as User Acceptance Testing (UAT) or Customer Acceptance Testing (CAT).

Acceptance testing is performed after the completion of the development and testing phase and before the deployment of the software. It is a crucial step in the software development life cycle, as it ensures that the software meets the user's expectations and requirements. Acceptance testing can be performed manually or automatically using testing tools.

The acceptance testing process begins with the preparation of test cases and scenarios based on the user's requirements and specifications. These test cases and scenarios are designed to simulate the real-world usage of the software and cover all possible scenarios that the user might encounter.

The acceptance testing process can be divided into three stages:

1. Preparation: In this stage, the test plan and test cases are prepared based on the user requirements and specifications. The test plan includes the objectives, scope, and approach of the acceptance testing. The test cases include the input data, expected results, and actual results.
2. Execution: In this stage, the test cases are executed by the testers or end-users, depending on the testing approach. The testers or end-users verify the functionality, usability, performance, and compatibility of the software against the requirements and specifications. Any defects or issues identified during the testing are reported and tracked until they are resolved.
3. Closure: In this stage, the test results are analyzed, and the software is accepted or rejected based on the acceptance criteria. The acceptance criteria are defined in the user requirements and specifications and represent the minimum level of quality and functionality that the software must meet to be accepted. If the software meets the acceptance criteria, it is approved for deployment to the end-users or customers. If the software does not meet the acceptance criteria, it is returned to the development team for further refinement.

Acceptance testing is important because it provides assurance to the end-users or customers that the software meets their requirements and expectations. It also helps to minimize the risks and costs associated with software development by detecting and resolving defects early in the development life cycle. Additionally, acceptance testing helps to improve the quality of the software by providing feedback to the development team for future improvements.

## Test-driven development:

Test-driven development (TDD) is a software development approach that emphasizes writing automated tests before writing production code. It involves a cycle of writing a failing test, writing code to make that test pass, and then refactoring that code to improve its design.

The TDD process typically follows these steps:

1. Write a failing test: Before writing any production code, write a test that fails. This test should reflect a specific requirement or behavior that the code is expected to implement.
2. Run the test: Run the test and confirm that it fails as expected. This step ensures that the test is valid and that it accurately reflects the desired behavior.
3. Write production code: Write just enough production code to make the test pass. The code should be written to satisfy the requirements of the test, without worrying about future requirements or code reusability.
4. Run the test again: Run the test again to ensure that the production code has solved the problem and that the test now passes.
5. Refactor the code: Once the test passes, refactor the production code to improve its design and reduce technical debt. This step involves making the code more modular, readable, and maintainable, while still keeping the tests passing.
6. Repeat: Repeat this cycle for the next requirement or behavior until the code is complete.

TDD is often used in conjunction with other development practices such as continuous integration and continuous delivery, which allow developers to automate the testing and deployment of code changes. This approach provides rapid feedback and encourages developers to write code that is more testable and maintainable.

TDD has several benefits, including:

* Ensuring that code is written to satisfy specific requirements or behaviors
* Encouraging developers to think about design and maintainability upfront
* Catching defects earlier in the development process
* Facilitating refactoring and reducing technical debt
* Encouraging the creation of modular and testable code
* Providing confidence in the correctness of the code

## Chosen testing methods:

After considering various testing methods commonly used, I have decided to conduct unit tests. These tests were focused on the software's functional requirements to ensure each one was thoroughly covered.

## Performed unit tests:

As for the actual testing process, since I lack automated unit tests, I performed all the tests on my own.

# Results and conclusions

## Conclusions drawn and lessons learned from papers examined:

A fresh approach to reward shaping in the context of reinforcement learning for text-based adventure games is proposed in the paper "Potential-based reward shaping for learning to play text-based adventure games."

During training, the authors present a potential-based reward shaping (PBRS) strategy that makes use of expert knowledge to provide additional rewards. There are two main components to the PBRS strategy: a potential function that measures the agent's progress toward the goal and a shaping function that gives the potential function additional rewards.

The authors evaluate and contrast their method with several baseline models using text-based adventure games. Their findings demonstrate that the PBRS strategy outperforms the baselines in terms of success rate and efficiency, demonstrating its ability to effectively utilize expert knowledge to enhance learning.

An examination of the potential and shaping functions is also included in the paper, which demonstrates that they are capable of capturing significant aspects of the game state and directing the agent toward the goal. The authors conclude that their method has the potential to make learning in dynamic and complex environments more effective and efficient.

A novel method for using deep reinforcement learning (DRL) to play text-based adventure games is described in the paper "Playing Text-Adventure Games with Graph-Based Deep Reinforcement Learning."

Using a graph convolutional network (GCN) to learn a representation of the state, the authors propose a graph based DRL method that models the game state as a graph. The GCN is prepared utilizing a strategy inclination technique to boost the normal combined reward.

The authors evaluate and contrast their method with several baseline models using a variety of text-based adventure games. The graph-based DRL method outperforms the baselines in both success rate and efficiency, demonstrating that it can effectively learn to play text-adventure games, as shown by their findings.

An examination of the learned representations is also included in the paper, demonstrating that the GCN is capable of meaningfully encoding key aspects of the game state. The authors conclude that their method has the potential to make learning in text-adventure games, which are complex and dynamic environments, more efficient and effective.

An overview of the use of graph transformation as a formalism for software specification and programming is provided in the paper "Graph Transformation for Specification and Programming."

The authors introduce the fundamental ideas behind graph transformation, such as the use of algebraic graph transformation systems, graphs as data structures, and rewriting rules for graphs. Then, they talk about how software engineering tasks like software specification, design, verification, and implementation all benefit from graph transformations.

The modelling of software architectures, the specification and implementation of a communication protocol, and the transformation of legacy code are just a few of the case studies presented in the paper that demonstrate the practical application of graph transformation.

Graph transformation is a powerful and adaptable formalism for software engineering and has the potential to make complex software systems development more effective.

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidenceA sample screen from the game

The picture shown above is the current title screen, which is where the game starts.

## Lessons learned:

I can confidently say that I am more comfortable with how software architecture works, why This project taught me how to write and design graphs. I have a better understanding of how algorithms are written and implemented in a real project and am aware of how the one I used (Dijkstra’s algorithm) works to find nodes in a graph and traverses to the shortest path between them. I have acquired knowledge on how other different types of algorithms like breadth-first search and reinforcement learning work and have a better understanding of their intricacies. I already know how to test said algorithms and have a better awareness of the complexity that comes with the usage of test libraries. I find myself more confident in my ability to create a basic user interface, implement it into a product, and modify it as I please. The ability to use Swing more effectively and being aware of how to implement a graphical component ought to be beneficial in future endeavors. I realize there are a lot more features I could potentially implement. Adding a proper inventory system and enhancing the visual element of the program (creating drawings indicating the player’s location or adding more than just plain text in the program would be a great first step). One of the most notable features I would like to put in the program is randomness. In a perfect world, I would have implemented a better, more nuanced combat system, would have strived to add random checks for passing skill checks that would fail if the player was below a certain level and would have created an algorithm that would work for an infinite dungeon type area. Testing more and in a noticeably reliable fashion would also greatly benefit me. Although the project forbids the usage of any additional libraries, I would be more than open to exploring some, so that I can enhance my project further. The alignment of elements (like panels, labels, etc.) within the project could be improved, because using the default coordinating system is clunky at best. After undertaking such an amount of work, I feel inspired for what is to come and will be doing my best to better and optimize this project to the best of my ability after I have a relatively finished version to present for the end of the semester. I know the problem being solved may not be as significant and important at first glance. Making the goal of this project more important is perhaps not the most achievable if tasks given the nature of the program, but upscaling it in the future can be done more easily.

## References:

https://aclanthology.org/N19-1358.pdf

<https://openreview.net/pdf?id=ZdysSJGubL>

<https://www.sciencedirect.com/science/article/pii/S0167642398000239>

<https://www.softwaretestinghelp.com/what-is-integration-testing>

Professor Anton Stoilov’s slides on AI algorithms

Professor Vladimir Georgiev’s slides on graphs from FDS

<https://www.techtarget.com/searchsoftwarequality/definition/unit-testing>

<https://www.javatpoint.com/java-swing>