Software Engineering 2: Project Teamwork
DETAILED DESIGN DESCRIPTION
A visual simulator for high-quality planning domain models

Mälardalen University

Group 2

January 11, 2019

Contents

1 Introduction 2

1.1 Client Background . . . 2

1.2 Definitions . . . 2

2 System Description 3

2.1 Use Case Diagram . . . 3

2.1.1 Actors . . . 4

2.1.2 Use Case 1: Load domain file and problem instance . . . 4

2.1.3 Use Case 2: View the current state . . . 4

2.1.4 Use Case 3: View state tree . . . 4

2.1.5 Use Case 4: Select an action . . . 5

2.1.6 Use Case 5: Execute an action . . . 5

2.1.7 Use Case 6: Go back to the last action . . . 5

2.2 High-level Description . . . 5

3 Software Architecture 6

3.1 Overview and Rationale . . . 6

3.2 Hardware/Software Mapping . . . 6

3.3 Error Handling . . . 6

4 Software Design Description 6

4.1 System Components (GUI) . . . 6

4.1.1 Component Interface Description . . . 6

4.1.2 Component Design . . . 7

5 Graphical User Interface 8

5.1 Overview of the User Interface . . . 8

5.2 Screen Images . . . 8

List of Figures

1 Use Case diagram . . . 3

2 Class diagram . . . 7

3 Screen Image 1 . . . 9

4 Screen Image 2 . . . 9

5 Screen Image 3 . . . 10

6 Screen Image 4 . . . 10

7 Screen Image 5 . . . 11

8 Screen Image 6 . . . 11

1 Introduction

1.1 Client Background

Ericsson is a Swedish company dedicated to the development of projects related to the area of telecommunications, highlighting in mobile technology, artificial intelligence applications and communication on the network.
The goal of this project, communicated to us from our client contact Swarup, is to create a tool for Ericsson to use when participating in project SCOTT. This project, in collaboration with other big companies like Bosch or Sonic, aims to take advantage of cross-domain integration in various industrial fields by strengthening Europe's position in technology and intelligent infrastructure processes. Ericsson is working on a use case of an automated warehouse where Artificial Intelligence planning is used to plan tasks for robots. To make this possible, we make use of PDDL (Planning Domain Definition Language), which is a successful attempt to standardize planning domain and problem description languages in AI planning.
The domain definition contains the domain predicates and operators (called actions). It may also contain types, constants and static facts. Since the domain definition is critical for a correct plan generation, it is necessary for our client to have a tool that visualizes these domain models.

1.2 Definitions

The language required by Ericsson to define domain models that are used as input to the planner is PDDL, a widely accepted language and considered a standard for domain specification and planning problems.
PDDL was first introduced in 1998 and was an attempt to standardize AI planning languages. It consists of two parts: the Domain definition and the problem definition.
The basic form of the **Domain Definition** contains logical facts called predicates, and a list of actions that can change the state of the model. Each predicate of the Domain definition corresponds objects that are defined in the Problem definition. An action in the Domain definition can be executed only if its preconditions are satisfied. The result of an action can change the value of the predicates.
The **Problem Definition** contains the objects of the modelled world, the initial state of the model and the desired goal state.

2 System Description

2.1 Use Case Diagram

The use case diagram is include in the Figure 1

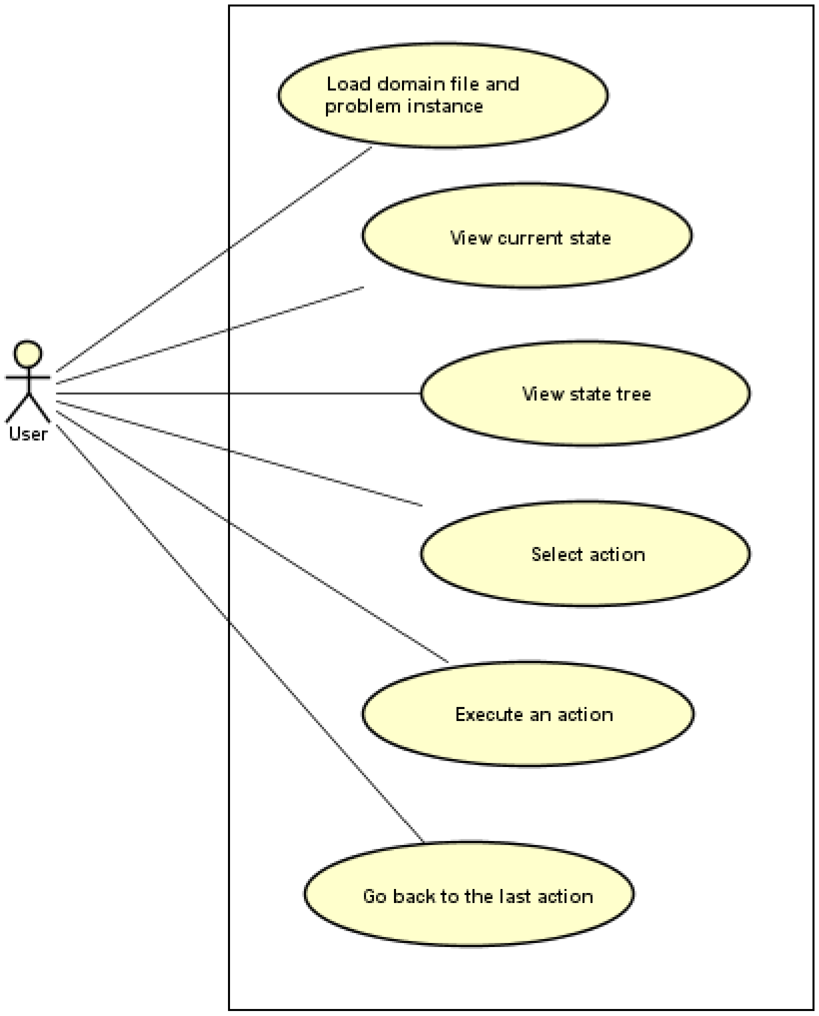


Figure 1: Use Case diagram

2.1.1 Actors

The only actors in this application will be the user and the system itself.
The **User** will be in charge of loading the PDDL files and simulating them. The **System** will be in charge of parsing and synthesizing these documents, as well as allowing the user to visualize the domain model and the status tree. In future updates, it would also check the syntax and edit PDDL files.

2.1.2 Use Case 1: Load domain file and problem instance

Name: Import
Initiator: User
Goal: Import the domain file and problem instance to edit or simulate/debug.
Main Scenario:
1. User import the domain file
2. problem instance

Extensions:
Imported files are in the wrong format:
a) The system throws an ERROR
b) Resume at 1

2.1.3 Use Case 2: View the current state

Name: View the current state
Initiator: User
Goal: The user selects the current state, being able to see how all the predicates are in that moment
Main Scenario:
1. The user selects the current state
2. The system shows the most important predicates for that state
3. By clicking a button, the user is able to see all the others states and predicates in the problem

2.1.4 Use Case 3: View state tree

Name: View state tree
Initiator: User
Goal: The user should be able to see the hole state tree of the states he has been throw.
Main Scenario:
1. The user selects the option of viewing the state tree
2. The state tree appears with all the states the user has been throw and all the actions he has been able to choose. If the state tree is bigger than the screen, there will be a scroll bar in the right side of the screen.
3. The user is able to choose a state in the state tree so he can view it

2.1.5 Use Case 4: Select an action

Name: Select an action
Initiator: User
Goal: The user chooses an action from the state tree.
Main Scenario:
1. The state tree appears with the position of the user in it
2. The user selects an action between the options he can take from his position
3. The system visualizes the selected state

2.1.6 Use Case 5: Execute an action

Name: Execute an action
Initiator: User
Goal: The user executes an action
Main Scenario:
1. The user selects an action from the state tree
2. The user executes the action previously selected
3. All the attributes from the first actions that are different than the ones in the action selected change, so the current state is updated

2.1.7 Use Case 6: Go back to the last action

Name: Go back to the last action
Initiator: User
Goal: The user goes back to the last current state, undoing the action made.
Main Scenario:
1. The user previously selects an action
2. The user selects the button to go back, so he can return to the last current state, and it is not recorded in the state history

2.2 High-level Description

The project aims to build a tool where the user can either import or edit a textual PDDL domain model, and simulate it for debugging purposes.
The tool that will be developed will provide a visual and textual simulation GUI that simplifies the debugging process by presenting the current state and the actions that are possible. The visual simulation of the states provides the user with an overview of how any potential deadlocks stages have occurred. When selecting one of the possible actions, the state should be updated. The tool should also provide the functionality of returning to a previous state, and resume the debugging from there.

3 Software Architecture

3.1 Overview and Rationale

This project uses the .NET framework created by Windows, whose aim is to develop applications and systems independent of the physical architecture and the operating system on which they are executed. It enables fast and secure application development and is highly adaptable when creating applications for various platforms. It includes several programming environments including C#, Visual Basic or F#.
The programming language we are using to carry out this project is C#, an object-oriented language. The client gave us the freedom to choose the programming environment, so we chose C#, as it is a language that most of the team controls and is quite simple to use and learn.

3.2 Hardware/Software Mapping

The tool in overall will be compiled and developed in Visual Studio, we choose visual studio because it is a good environment and most of the group members have worked with it. The tool itself will be run on Windows and only windows. The requirements that the system running the tool should have is basic software requirements because the tool we are developing is a basic level version and don't need some advanced software to be able to run the tool.

3.3 Error Handling

While testing the project errors encountered will be documented as cards on Trello in the priority queue.
Developers will be taking the tasks off the top of the queue.
Once the problem has been solved they will move onto the next problem.
When the project is released errors will be filed through issues on GitHub from end users.
From there on they will be transferred to Trello if deemed proper.

4 Software Design Description

4.1 System Components (GUI)

4.1.1 Component Interface Description

This system component is what the user sees and interacts with when using the tool. Graphical components such as the tool window, buttons, views and textboxes are implemented using WinForms which is a graphical class library included in the Microsoft .NET framework.

4.1.2 Component Design

As the tool is a WinForm application, the flow of the application will be event-driven determined by events such as user actions (mouse clicks). The logic behind each GUI component is implemented using C#. The backend logic consists of 6 classes, State, StateManager, GUIManager, Domain, Problem, Action, Predicate, shown in the Figure 2.

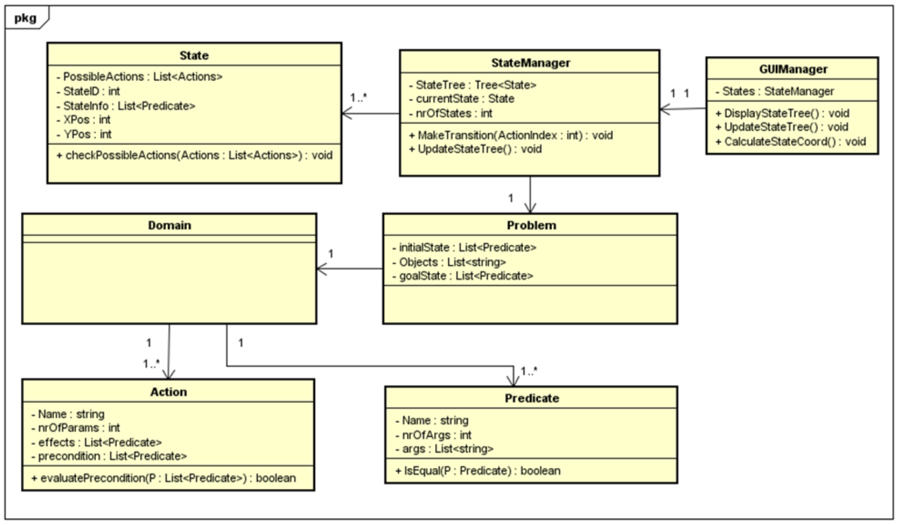

pkg
State
- PossibleActions : List<Actions>
- StateID : int
- StateInfo : List<Predicate>
- XPos : int
- YPos : int
+ checkPossibleActions(Actions : List<Actions>) : void
1..*
StateManager
- StateTree : Tree<State>
- currentState : State
- nrOfStates : int
+ MakeTransition(ActionIndex : int) : void
+ UpdateStateTree() : void
1 1
GUIManager
- States : StateManager
+ DisplayStateTree() : void
+ UpdateStateTree() : void
+ CalculateStateCoord() : void
1
Domain
1
Problem
- initialState : List<Predicate>
- Objects : List<string>
- goalState : List<Predicate>
1
1..*
1
1..*
Action
- Name : string
- nrOfParams : int
- effects : List<Predicate>
- precondition : List<Predicate>
+ evaluatePrecondition(P : List<Predicate>) : boolean
Predicate
- Name : string
- nrOfArgs : int
- args : List<string>
+ IsEqual(P : Predicate) : boolean

Figure 2: Class diagram

Description of the classes:

- Domain, Problem, Action and Predicate are used to represent the PDDL domain and problem definitions in C# datatypes.
- State class represent the nodes in the state tree where all the information such as the position of the state in the tree, predicates. While the StateManager class keeps track of the structure of the whole state tree and where the agent/user currently is located in the tree. When the user wants to make a transition to another state, it is the StateManager class that is responsible for moving the current state of the user and updating the state tree and possible actions.
- GUIManager class handles the visual aspects of the state tree by drawing the objects on the WinForm window. To do this, the GUIManager retrieves necessary data from the StateManager class.

5 Graphical User Interface

Visualization is the most important part of this project.
We aim to provide users with an intuitive Graphical User Interface (GUI).

5.1 Overview of the User Interface

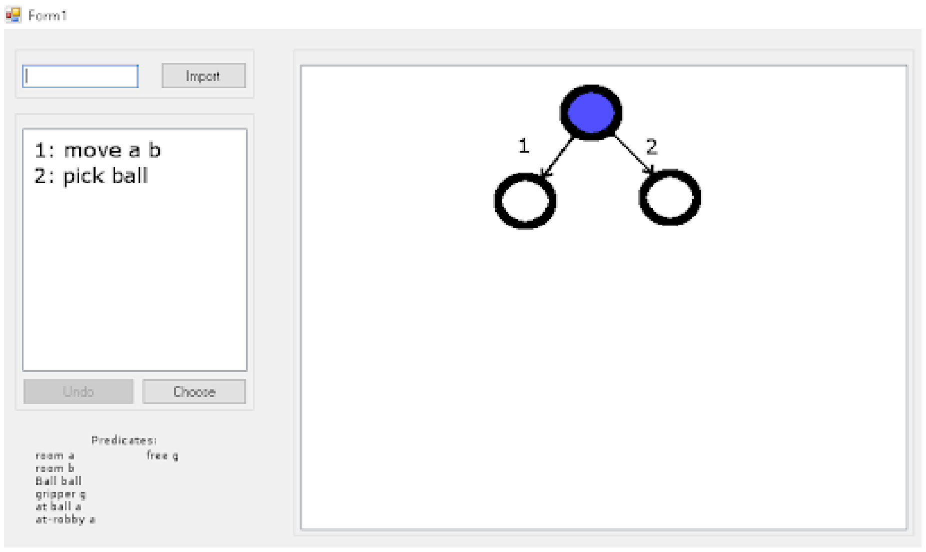
The GUI will be divided into four parts; import, actions, description and state tree.
The **import** section of the screen (top left) will include an import button and a text field where the absolute path to the code should be written in.
The **actions** portion of the screen (middle left) will contain information about the available actions and buttons to do and undo those actions.
The **description** (bottom left) will be a log box with a detailed description of the current state.
Finally, the state tree (right) will provide a **state tree** with all the possible states.

The states will be in different colours making it easier to distinguish them:

- blue: current state
- green: next state (while taking an action)
- orange: deadlock state
- grey: all other states

5.2 Screen Images

PDDL file is imported. List of all the possible actions is shown in the middle left part. Description of the state is shown in the bottom left part. State tree with the current state coloured blue is shown on the right. (Figure 3)
The first action is chosen, which changes the current state and shows only the "undo" action which will take us to the previous state. (Figure 4)
The only possible action is chosen which brings us back to the initial state. (Figure 5)
The second action is taken which takes us to the new state. The two new actions are shown, "undo" action and an action that will take us to the new state. (Figure 6)
The new action is taken and the state tree updates. (Figure 7)
We reached the goal state, there are no more actions to be taken. (Figure 8)

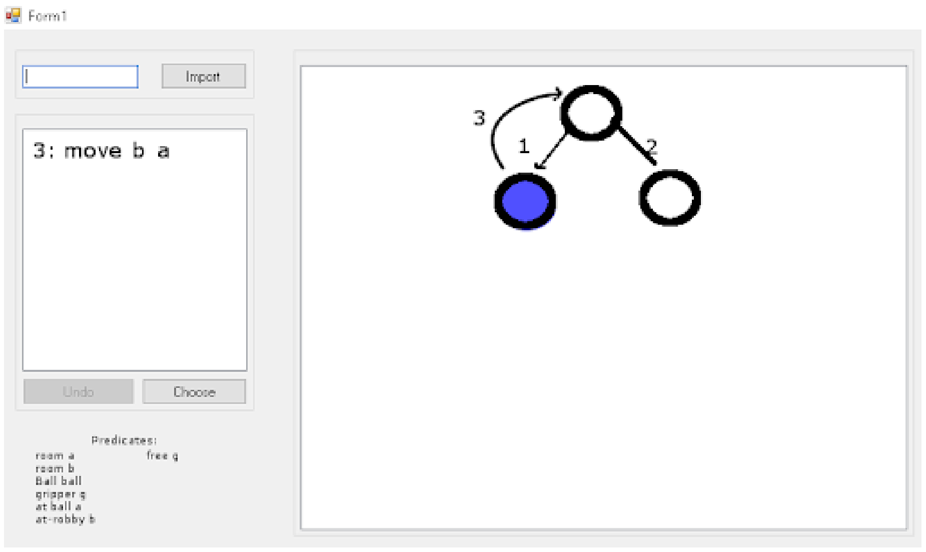


1: move a b
2: pick ball

Predicates:

```
room a                 free g
room b
Ball ball
gripper g
at ball a
at-robby a
```

Figure 3: Screen Image 1

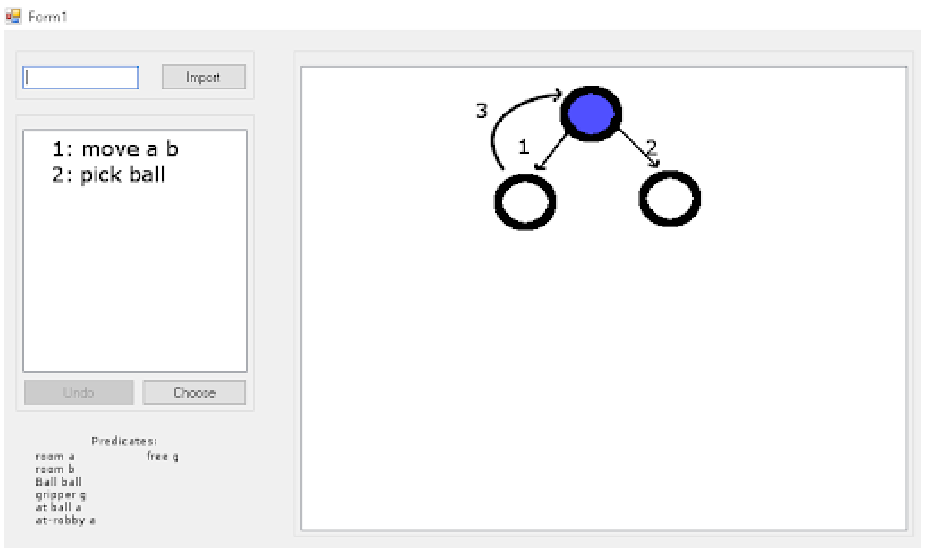


3: move b a

Predicates:

```
room a                 free g
room b
Ball ball
gripper g
at ball a
at-robby b
```

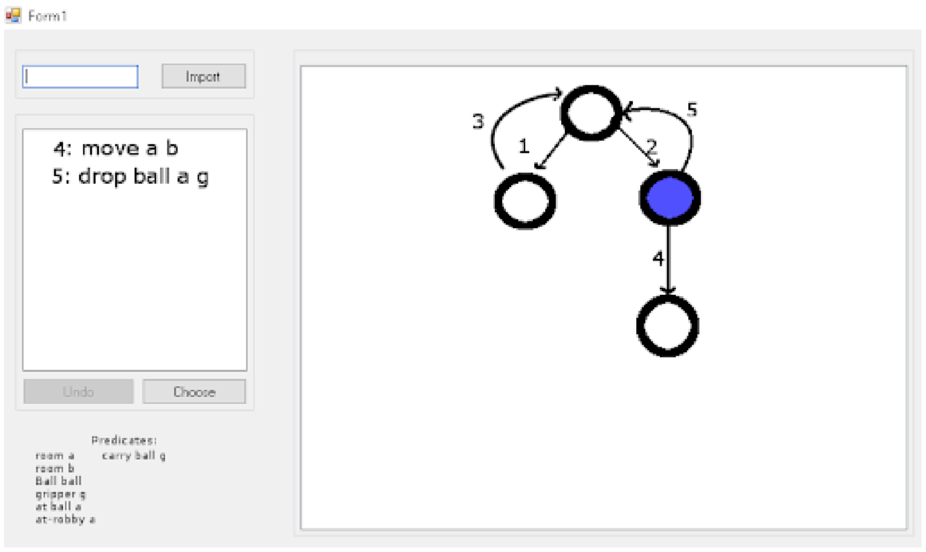
Figure 4: Screen Image 2



1: move a b
2: pick ball

```
          Predicates:
room a            free g
room b
Ball ball
gripper g
at ball a
at-robby a
```

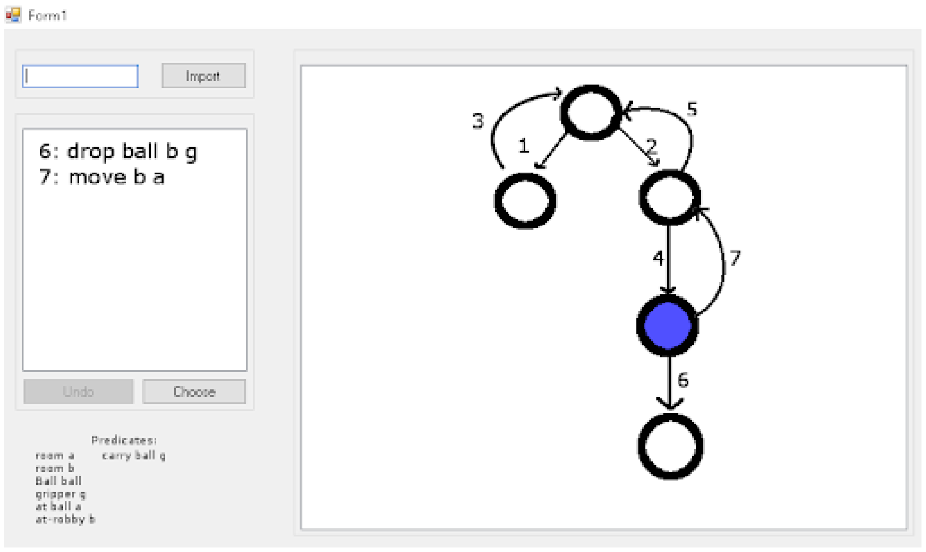
Figure 5: Screen Image 3



4: move a b
5: drop ball a g

```
          Predicates:
room a      carry ball g
room b
Ball ball
gripper g
at ball a
at-robby a
```

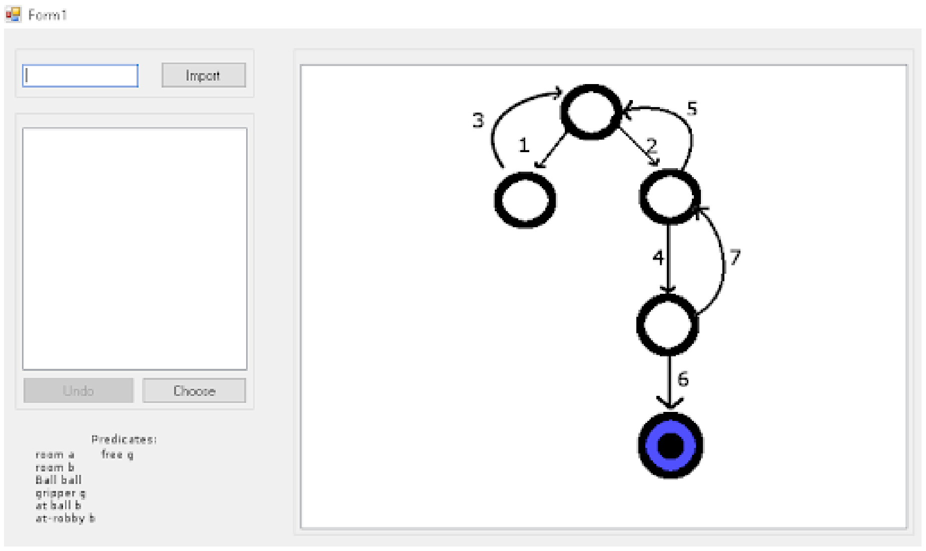
Figure 6: Screen Image 4



```
6: drop ball b g
7: move b a
```

```
           Predicates:
room a       carry ball g
room b
Ball ball
gripper g
at ball a
at-robby b
```

Figure 7: Screen Image 5



```
           Predicates:
room a       free g
room b
Ball ball
gripper g
at ball b
at-robby b
```

Figure 8: Screen Image 6