Artificial Intelligence

Winter 2016-2017

HW2

STRIPS

Students:
Ilya Holtz: 319236931
Volodymyr Dzhuranyuk: 323792796
Yuval Bernshtein

Lecturer: Prof. Larry Manevitz

Submission date: 29.01.2017

Source code and executable:
- On the usb drive in the attached envelope
- Executable: usbDisk/exec/Strips.jar
- On Github public repository: ***https://github.com/holtzilya2008/stripsAI***

Overview

This document will describe the Implementation of the STRIPS algorithm, for solving the “moving furinture problem”. First we will describe our definition of Actions and Conditions, and then we will talk about the heuristic in witch we are adding new goals to our Goal stack.

Most of the document we will use well documented samples of our code.

Document structure

1. The problem - Brief description
2. STRIPS - Definitions: Problems, Actions and Conditions
3. STRIPS - The Engine - Main Idea
4. The obstacle case
5. The different room case
6. Technical overview
7. Installation and using Guide

1. The problem - Brief descriprion

We have a a board of 12x20, in it's initial state and final state. In the Initial state, we have rectangular furniture placed on the board. In the final place, we have the same furniture placed on the board, but in different position.

Our goal is to make a plan, in witch we take actions, to reach from the initial state of the board to the final state. Which means, we have to move each furniture, to its new place in the final state.

We can preform only actions from a predefined list. And we must define a list of conditions, on witch our heuristic will be based.

2. STRIPS - Definitions: Problems, Actions and Conditions

Our Actions (Defined in the assignment document):

CODE: ACTIONS DEFINITIONS : from Action.java

```
      private static final String DEBUG_TAG = "class Action";
      private static final int DEBUG_ALL = 0;
      private static final int DEBUG_CLASS = 1;
      private static final int DEBUG_FUNCTION = 2;
      private static final int DEBUG_SPECIFIC = 3;
      private static final int CURRENT_DEBUG_LEVEL = DEBUG_ALL;
```

CODE: ACTIONs PRECONDITIONS : from Action.java

```
    private void setPreconditions(){
        ArrayList<RecInfo> args = new ArrayList<RecInfo>();
        args.add(furniture);
        Condition condition = null;
        switch(name){
            case MOVE_LEFT:
                condition = new Condition(api,
                                                    Condition.CAN_MOVE_LEFT,
                                                    args,
                                                    true);
                break;
            case MOVE_RIGHT:
                condition = new Condition(api,
                                                    Condition.CAN_MOVE_RIGHT,
                                                    args,
                                                    true);
                break;
            case MOVE_UP:
                condition = new Condition(api,
                                                    Condition.CAN_MOVE_UP,
                                                    args,
                                                    true);
                break;
            case MOVE_DOWN:
                condition = new Condition(api,
                                                    Condition.CAN_MOVE_DOWN,
                                                    args,
                                                    true);
                break;
            case ROTATE_LEFT:
                condition = new Condition(api,
                                                    Condition.CAN_ROTATE_LEFT,
                                                    args,
                                                    true);
                break;
            case ROTATE_RIGHT:
                condition = new Condition(api,
                                                    Condition.CAN_ROTATE_RIGHT,
                                                    args,
                                                    true);
                break;
```

```
            default:
                break;
        }
        preconditions.add(condition);
    }
```

Conditions definition:

CODE: OUR CONDITIONS : from Condition.java

```
      public static final String IN_PLACE = "IN_PLACE";
      public static final String CAN_MOVE_UP =  "CAN_MOVE_UP";
      public static final String CAN_MOVE_DOWN = "CAN_MOVE_DOWN";
      public static final String CAN_MOVE_LEFT = "CAN_MOVE_LEFT";
      public static final String CAN_MOVE_RIGHT = "CAN_MOVE_RIGHT";
      public static final String CAN_ROTATE_RIGHT = "CAN_ROTATE_RIGHT";
      public static final String CAN_ROTATE_LEFT = "CAN_ROTATE_LEFT";
      public static final String ROTATED = "ROTATED";
      public static final String IN_SPACE = "IN_SPACE";
      public static final String IS_LOWER = "IS_LOWER";
      public static final String IS_HIGHER = "IS_HIGHER";
      public static final String IS_TO_THE_LEFT = "IS_TO_THE_LEFT";
      public static final String IS_TO_THE_RIGHT = "IS_TO_THE_RIGHT";
```

Each condition can be true or false. Each condition has a DesiredValue and Check() Method. The Check() method, checks the condition in the “Real World”.
If The condition is satisfied, then we must have: Check() == desiredValue;

CODE: isSatisfied() method : from Condition.java

```
      /**
       * Method Checks if the condition is satisfied according to our
       * desired value.
       * @return Check() AND desired value
       */
      public boolean isSatisfied(){
            validateArgs();
            return (desiredValue == Check());
      }
```

A Problem definition:
In this Project, a “Problem” is when we have source Rectangle, and we have a targed place, that it needs to reach:

CODE: Class Problem constructor from Problem.java

```
      public Problem(RecInfo source, RecInfo targed){
            this.source = source;
            this.targed = targed;
      }
```

3. STRIPS - The Engine : The main Idea

We have a GoalStack, initialized with IN_PLACE conditions (Our problems)
The IN_PLACE condition has two arguments, source and targed. We have to place the source Rectangle in the targed place. The following code is our Solve function, It is well documented, so it can explain our main idea.

CODE: OUR SOLVE METHOD: from StripsEngine.java

```
    public void Solve(){
        debugPrint(DEBUG_FUNCTION,"reached Solve");
        iterationsSinceLastMove = 0;
        movesMade = 0;
        prepareGoalStack();

        /* We will continue to solve problems until our goalStack is empty */
        while(goalStack.empty() != true){
            //Check if we are not in an infinite loop:
            if(isInfiniteLoop()){
                break;
            }

            // Lets look at our top Goal, if it is satisfied, just pop it out
            iterationsSinceLastMove++;
            Condition currentGoal = goalStack.peek();
            Condition currentProblem = problemStack.peek();
            debugPrint(DEBUG_FUNCTION,
                "currentGoal is "+currentGoal.toString() + "\n" +
                "currentProblem is "+currentProblem.toString());
            if(currentGoal.isSatisfied()){
                debugPrint(DEBUG_FUNCTION,"The Goal:"+currentGoal.toString() +
                        "poped out of the GoalStack");
                if(currentGoal == currentProblem){
                    problemStack.pop();
                }
                popFromGoalStack();
            }else{
                switch(currentGoal.getName()){
                    /* If the new Goal is IN_PLACE type and it is not satisfied,
                     * Then we will add SubGoals to our GoalStack
                     */
                    case Condition.IN_PLACE:
                        /* If we face a new problem that we never seen before
                         * add it to the problemStack
                         */
                        if(currentGoal != currentProblem){
                            problemStack.push(currentGoal);
                        }
                        /* Adding 6 new Conditions to the GoalStack
                            ST-5.   IS_TO_THE_LEFT(rect1,targed1) = false
                            ST-4.   IS_TO_THE_RIGHT(rect1,targed1) = false
                            ST-3.   IS_LOWER(rect1,targed1) = false
                            ST-2.   IS_HIGHER(rect1,targed1) = false
                            ST-1.  ROTATED(rect1,targed1) = false - Both must not
                                be rotated one related to another
                            ST.  IN_SPACE(source,ROOM(targsed)) = true - They must
                                in the same room
                         */
                        ArrayList<RecInfo> furniture = currentGoal.getArgs();
                        Condition newGoal = new Condition(api,
                                                Condition.IS_TO_THE_LEFT,
                                                          furniture,false);
```

```
                    pushToGoalStack(newGoal);
                    newGoal = new Condition(api,
                                            Condition.IS_TO_THE_RIGHT,
                                                    furniture,false);
                    pushToGoalStack(newGoal);
                    newGoal = new Condition(api,
                                            Condition.IS_LOWER,
                                                    furniture,false);
                    pushToGoalStack(newGoal);
                    newGoal = new Condition(api,
                                            Condition.IS_HIGHER,
                                                    furniture,false);
                    pushToGoalStack(newGoal);
                    newGoal = new Condition(api,
                                                    Condition.ROTATED,
                                                    furniture,false);
                    pushToGoalStack(newGoal);
                    RecInfo targed = furniture.get(1);
                    RecInfo desiredRoom = api.getRoom(targed);
                    ArrayList<RecInfo> args = new ArrayList<RecInfo>();
                    args.add(furniture.get(0));
                    args.add(desiredRoom);
                    newGoal = new Condition(api,
                                            Condition.IN_SPACE,
                                                    args,true);
                    pushToGoalStack(newGoal);
                    break;

                /* If we reach the following cases, it means we try to
                 * apply an Action, but we ran in to an Obstacle.
                 * We try first to avoid dealing with the Obstacle using:
                 * canAvoidObstacle() method but if we can't avoid it,
                 * we choose to move the Obstacle from our way using
                 * handleObstacleCase() Method.
                 * In each case bellow, we must pop the CAN_MOVE_...
                 * condition type from the GoalStack, before continue.
                 */
                case Condition.CAN_MOVE_UP:
                    popFromGoalStack();
                    if (!canAvoidObstacle(Action.MOVE_UP)) {
                        handleObstacleCase(Action.MOVE_UP);
                    }
                    break;
                case Condition.CAN_MOVE_DOWN:
                    popFromGoalStack();
                    if (!canAvoidObstacle(Action.MOVE_DOWN)) {
                        handleObstacleCase(Action.MOVE_DOWN);
                    }
                    break;
                case Condition.CAN_MOVE_LEFT:
                    popFromGoalStack();
                    if (!canAvoidObstacle(Action.MOVE_LEFT)) {
                        handleObstacleCase(Action.MOVE_LEFT);
                    }
                    break;
                case Condition.CAN_MOVE_RIGHT:
                    popFromGoalStack();
                    if(!canAvoidObstacle(Action.MOVE_RIGHT)){
                        handleObstacleCase(Action.MOVE_RIGHT);
                    }
                    break;
                case Condition.CAN_ROTATE_RIGHT:
                    popFromGoalStack();
                    handleObstacleCase(Action.ROTATE_RIGHT);
```

```
                    break;
                case Condition.CAN_ROTATE_LEFT:
                    popFromGoalStack();
                    handleObstacleCase(Action.ROTATE_LEFT);
                    break;
                case Condition.IN_SPACE:
                    moveBetweenRooms();
                    break;

                /* Here we start to deal with Actions
                 * see handleActionCase() method below
                 */
                case Condition.ROTATED:
                    RecInfo source = currentGoal.getArgs().get(0);
                    if(api.CanRotateRight(source)){
                        handleActionCase(Action.ROTATE_RIGHT);
                    }else{
                        handleActionCase(Action.ROTATE_LEFT);
                    }
                    break;
                case Condition.IS_LOWER:
                    handleActionCase(Action.MOVE_UP);
                    break;
                case Condition.IS_HIGHER:
                    handleActionCase(Action.MOVE_DOWN);
                    break;
                case Condition.IS_TO_THE_LEFT:
                    handleActionCase(Action.MOVE_RIGHT);
                    break;
                case Condition.IS_TO_THE_RIGHT:
                    handleActionCase(Action.MOVE_LEFT);
                    break;
                default:
                    break;
            }   //switch
        }   //else
    } // while
    printPlan();
}
```

4. The Obstacle case

Bellow you can see the function which deals with the obstacle case:

CODE: THE OBSTACLE CASE: from StripsEngine.java

```
   /**
     * Method to handle an Obstacle case
     * If we reached thease cases, means that we want to
     * apply an action, but we have an Obstacle. In the
     * current engine, we choose to move the Obstacle aside
     * to let us apply the action and after we apply it, we
     * return the obstacle to it's original place
     * so we add the following to the GoalStack:
     * ST-2. IN_PLACE(obstacle,oldObstaclePlace) = true : Return the
     *  obstacle to it's original place
     * ST-1. IN_PLACE(source, targed) = true :
     *  move source to targed
     * ST. IN_PLACE(obstacle,tmpPlace) = true : Move the
     *  obstacle to temprary place
     */
    private void handleObstacleCase(String action){
        Condition currentProblem = problemStack.peek();
        Condition currentGoal = goalStack.peek();
        ArrayList<RecInfo> args;
        Condition newGoal;
        RecInfo source = currentProblem.getArgs().get(0);
        RecInfo targed = currentProblem.getArgs().get(1);
        RecInfo obstacle = api.getObstacle(source,
                                    action);
        //Rotation and walls case
        if(currentGoal.getName() == Condition.ROTATED && obstacle == null){
            RecInfo tmpRotationPlace = null;
            if(action == Action.ROTATE_LEFT){
                tmpRotationPlace = api.findPlaceForRotationLeft(source);
            }else{
                tmpRotationPlace = api.findPlaceForRotationRight(source);
            }
            if(tmpRotationPlace == null){
                Global.InvokeAssert(true,"Engine: ERROR! tmpRotationPlace is null");
            }
            args = new ArrayList<RecInfo>();
            args.add(source);
            args.add(tmpRotationPlace);
            newGoal = new Condition(api,
                                        Condition.IN_PLACE,
                                        args,true);
            pushToGoalStack(newGoal);
            return;
        } // Rotation and walls case

        debugPrint(DEBUG_SPECIFIC,"getObstacle returned "+obstacle);
        RecInfo oldObstaclePlace = obstacle.copy();
        oldObstaclePlace.setDummy();
        RecInfo tmpPlace = api.findTempObstaclePlace(source,
                                    obstacle,
                                    action);
        debugPrint(DEBUG_SPECIFIC,"findTempObstaclePlace returned "+tmpPlace);
        boolean needToReturnObstacle =
doWeneedToReturnObstacle(oldObstaclePlace,obstacle);

        if(needToReturnObstacle){
```

```
            args = new ArrayList<RecInfo>();
            args.add(obstacle);
            args.add(oldObstaclePlace);
            newGoal = new Condition(api,
                                            Condition.IN_PLACE,
                                            args,true);
            pushToGoalStack(newGoal);
            args = new ArrayList<RecInfo>();
            args.add(source);
            args.add(targed);
            newGoal = new Condition(api,
                                            Condition.IN_PLACE,
                                            args,true);
            pushToGoalStack(newGoal);
        }

        args = new ArrayList<RecInfo>();
        args.add(obstacle);
        args.add(tmpPlace);
        newGoal = new Condition(api,
                                        Condition.IN_PLACE,
                                        args,true);
        pushToGoalStack(newGoal);
    }
```

CODE: OBSTACLE AVOIDENCE from StripsEngine.java

```
    /**
     * This method will run in case something (an obstacle) is preventing
     * us to move in a certain direction, but we want to avoid touching it and
     * instead check other options for us to move.
     * @param : Action, witch specifies the direction we want to move in
     * @return : true if we can avoid the obstacle, else false
     */
    private boolean canAvoidObstacle(String action){
        Condition currentProblem = problemStack.peek();
        Condition currentGoal = goalStack.peek();
        RecInfo source = currentGoal.getArgs().get(0);
        RecInfo targed = currentProblem.getArgs().get(1);
        Condition newGoal;
        ArrayList<RecInfo> args = new ArrayList<RecInfo>();
        args.add(source);
        args.add(targed);
        switch(action){
            case Action.MOVE_LEFT:
            case Action.MOVE_RIGHT:
                if(api.IsHigher(source,targed) && api.CanMoveDown(source)){
                    newGoal = new Condition(api,Condition.IS_HIGHER,
                                                    args,false);
                    pushToGoalStack(newGoal);
                    return true;
                }
                if(api.IsLower(source,targed) && api.CanMoveUp(source)){
                    newGoal = new Condition(api,Condition.IS_LOWER,
                                                    args,false);
                    pushToGoalStack(newGoal);
                    return true;
                }
```

```
                break;
            case Action.MOVE_UP:
            case Action.MOVE_DOWN:
                if(api.IsToTheLeft(source,targed) && api.CanMoveRight(source)){
                    newGoal = new Condition(api,Condition.IS_TO_THE_LEFT,
                                                        args,false);
                    pushToGoalStack(newGoal);
                    return true;
                }
                if(api.IsToTheRight(source,targed) && api.CanMoveLeft(source)){
                    newGoal = new Condition(api,Condition.IS_TO_THE_RIGHT,
                                                        args,false);
                    pushToGoalStack(newGoal);
                    return true;
                }
                break;
            default:
                debugPrint(DEBUG_FUNCTION,"canAvoidObstacle(): BUG - Reached
default case");
        }
        return false;
    }
```

5. The different room case

If our source is in different rooms, we first move to good spot near the doorWay of the current room, then move through the door, and then move to the targed place.

CODE: moveBetweenRooms method from StripsEngine.java

```
  /**
     * Method adds Goals for moving the source furniture to the specified room
     * - currentDoorway: we find a spot near the doorway in current room
     * - targedDoorway: we find a spot near the doorway in the target room
     * - we add the following to the GoalStack:
     * ST-4 : IS_TO_THE_LEFT(source,targedDoorway) = false
     * ST-3 : IS_TO_THE_RIGHT(source,targedDoorway) = false
     * ST-2 : IS_LOWER(source,targedDoorway) = false
     * ST-1 : IS_HIGHER(source,targedDoorway) = false
     * ST : IN_PLACE(source,currentDoorway)
     */
    private void moveBetweenRooms(){

        Condition currentGoal = goalStack.peek();
        RecInfo source = currentGoal.getArgs().get(0);
        RecInfo currentRoom = api.getRoom(source);
        RecInfo targedRoom = currentGoal.getArgs().get(1);
        RecInfo currentDoorway = api.findSpotNearDorway(source,
                                                        currentRoom,
                                                        targedRoom);
        RecInfo targedDoorway = api.findSpotNearDorway(source,
                                                        targedRoom,
                                                        currentRoom);
        ArrayList<RecInfo> args = new ArrayList<RecInfo>();
        args.add(source);
        args.add(targedDoorway);
        Condition newGoal = new Condition(api,
                                          Condition.IS_TO_THE_LEFT,args,false);
        pushToGoalStack(newGoal);
        newGoal = new Condition(api,Condition.IS_TO_THE_RIGHT,args,false);
        pushToGoalStack(newGoal);
        newGoal = new Condition(api,Condition.IS_LOWER,args,false);
        pushToGoalStack(newGoal);
        newGoal = new Condition(api,Condition.IS_HIGHER,args,false);
        pushToGoalStack(newGoal);

        args = new ArrayList<RecInfo>();
        args.add(source);
        args.add(currentDoorway);
        newGoal = new Condition(api,Condition.IN_PLACE,args,true);
        pushToGoalStack(newGoal);
    }
```

6. Technical overview and using Guide

We Implemented our project using Java language. We took the GUI from Jonathan Liberman & Itay Segev and modified it according to our needs. To the original GUI we added Multithreading and Timing for showing the rectangle transitons live and to controll the speed of each move.

1) Please Enter the desired move speed to the text field named: MS per move in milliseconds
2) Place the original rectangular furniture on the left board named “Current” and place corresponding rectangles on the right board named “What We want”
3) If you want to observe the Goal stack while the program is working, press on a button “Show Stack” at the top right corner, and place the stack window in any desired place on the screen.
4) Press “Start Simulator” button and enjoy :)

7. Installation

To run the executable Strips.jar file, you must have Oracle java 8 JRE installed on your machine.

You can find the executable Strips.jar file in /exec folder on the attached usb drive. or you can download it from the github repository on:
https://github.com/holtzilya2008/stripsAI.git
and enter the /exec folder in the repo.