Workflow of Subgoal A Agent*

Solving the rectangle track of Geometry Friends

Daniel Fischer
Department of Knowledge Engineering
Maastricht University
Maastricht, The Netherlands
daniel.fischer@student.maastrichtuniversity.nl

Supervisor: Dr. M. H. M. Winands
Department of Knowledge Engineering
Maastricht University
Maastricht, The Netherlands
m.winands@maastrichtuniversity.nl

***Abstract*—This report gives an overview about the developed Subgoal A* Agent to solve the rectangle track of the game Geometry Friends. The structure and functionality is briefly explained.**

I. Introduction

From September 2014 to March 2015 I worked on my master thesis. In the thesis the search techniques Monte-Carlo Tree Search (MCTS) and A* were investigated for Geometry Friends. The Subgoal A* Agent is the result of the thesis. For more and detailed information you can read my master thesis "DEVELOPMENT OF SEARCH-BASED AGENTS FOR THE PHYSICS-BASED SIMULATION GAME GEOMETRY FRIENDS". In the next section I describe the workflow of the agent.

II. Workflow

A. Abstraction

The abstraction represents the structure of a level and possible actions of Geometry Friends. This is the foundation to compute routes with the search technique Subgoal A*. This abstraction takes into account the rectangle player and its abilities. The approach of the abstraction is based on the following two articles. The article by Kim et al. [1] describes a directed graph abstraction for Geometry Friends. The graph consists of nodes and one/two-way edges. For example, nodes are at the ends of an obstacle, at a narrow alley or at the point where the player falls down. The edges describe how node B can be reached by node A if there is an arrow from node A to B. For instance, the rectangle player has three different edges for a "Square", "Horizontal rectangle" and "Vertical rectangle" action. Within this graph building process physical constraints are taken into account but the paper does not explain it and the physical properties of the game Geometry Friends are unknown. Next, the article by Perez et al. [2] describes an abstraction for the Physical Traveling Salesman Problem, which uses only the collectables. The approach of an own abstraction is to combine everything, the nodes given by the obstacles, create nodes for the collectables and take into account the physics of the game. The abstraction is computed when a level starts. This means the implementation is in the class *SquareAgent/RectangleAgent*. The abstraction is performed at the first call of the *Update* method. At this step all information is given. A first simple representation of the level is required to compute the nodes and edges of the abstraction. This representation is built by the given obstacle information. In a boolean matrix all obstacles are set as true and the free space is set as false. With this matrix the nodes and edges can be calculated.

Nodes are key pixels in the level to follow a route:

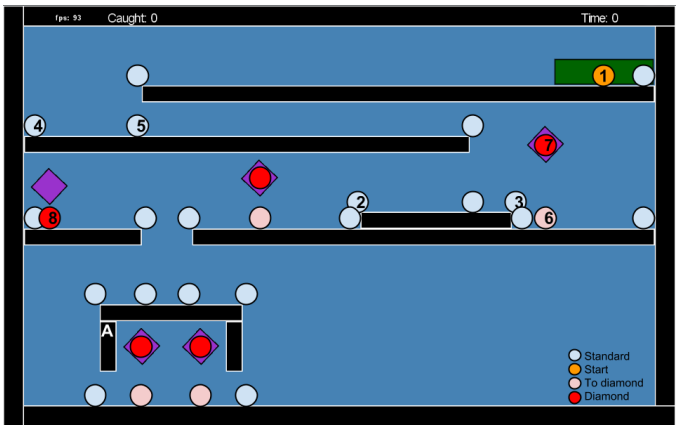

Fig. 1. Nodes of rectangle track 2013 level 5

The edges between two nodes describe the action the player has to perform to get from one node to another one. The edges are represented in an adjacency matrix with numbers from 0 to 3. The number 1 implies an action as a square, 2 an action as a horizontal rectangle, and 3 an action as a vertical rectangle. The number 0 implies there is no edge between the nodes. Another important piece of information is the direction the player has to follow to reach the goal node using the given action of the adjacency matrix. The direction is also stored in a matrix with the same size as the adjacency matrix. There are eight different directions from 0 to 7, from right to upper right, clockwise. The last piece of information, for later calculations, are the distances between nodes with an edge. The distances are also stored in a matrix. To compute the distance the Euclidean distance is used.

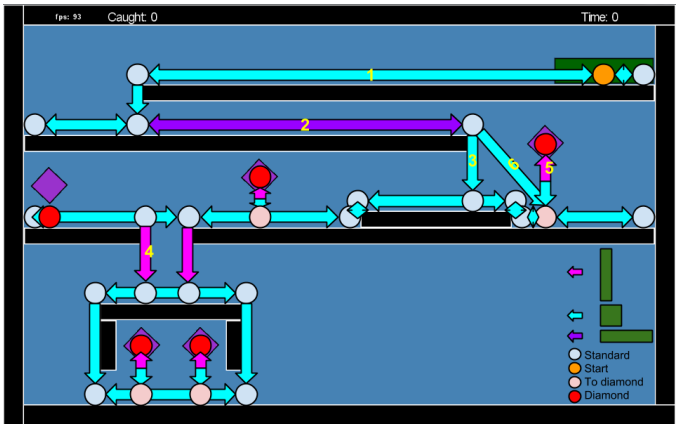

Fig. 2. Nodes and edges of rectangle track 2013 level 5

	1	2	3
1	0	0	1
2	0	0	3
3	1	0	0

(a) Actions

	1	2	3
1	0	7	0
2	3	0	2
3	4	6	0

(b) Directions

	1	2	3
1	0	484	63
2	484	0	480
3	63	480	0

(c) Distances

Fig. 3. Examples of three matrices

B. Search

After building the abstraction Subgoal A* is used to calculate the most sophisticated route.

This approach of A* does not use any predefined diamond order. It is an adapted A* algorithm where the A* node has a new property and the heuristic is set to 0. The property stores the collected diamonds and the order of the diamonds for each node. It is used to search for all diamonds with the ordinary A* pathfinding. An example is given with Figure 4. There are five nodes and two of them are diamond nodes. Without the new property A* cannot find both diamonds and would return only the route A, B, C or A, D, E, depending on which diamond will be collected last. With the new property there are different states of B, for instance, B without a diamond. B with diamond collected at C. B with diamond collected at E. These collected diamonds are stored in the new property. This means, if A* searches from A to B to C, A* will not stop but searches again back to B because the first B state does not contain the diamond collected at C and in the current state the diamond was collected at C. This is equal for all other states so that A* will lead to the diamond at E and returns a complete route from A to B to C to B to A to D to E. For complex levels it is possible that the search time is high so that the given number of diamonds to collect is decreased by one for instance every two seconds. It is also possible that in less than two seconds A* already finishes the search for all diamonds but could not find a route. Then the number of diamonds is also decreased by one and the algorithm starts again.

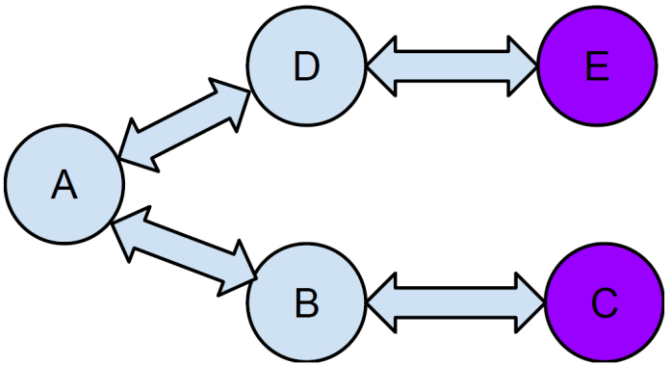

Fig. 4. Subgoal A* graph example

C. Driver

After calculating the route with the search algorithm the driver is required to follow this route in the level. The driver is created after the abstraction calculation and search. It needs the nodes, adjacency matrix, direction map of the abstraction and the route of the search technique. The *GetAction* method of the driver is performed in the *Update* method in the *SquareAgent/RectangleAgent* class. The rectangle executes the actions, which the driver returns. *GetAction* needs the current position of the rectangle and is called each time step (40 ms). Rules are used to check, which action should be executed next or whether a next node of the route is reached. The current position of the rectangle, the previous and next nodes, the previous and next actions, the previous and next directions, and the distance are variables to check which rule is applicable.

Two of 13 rules are:

- Rule 1 of the driver is to let the rectangle drive back if it gets stuck while driving to the pseudo node to fall down through a gap. Figure 5 shows the rectangle, which gets stuck. Rule 1 is used if the next node is a pseudo node and the first entry of the distance list and tenth entry of the distance list is the same. If the direction is to the right and the distance is lower than 200, to accelerate in direction left is returned. If the direction is to the left and the distance is lower than 200, to accelerate in direction right is returned.
- Rule 2 is to accelerate randomly to the left or right if the rectangle get stuck for 600 ms in a diagonal position. An example is shown in Figure 6. The diagonal position is checked by the method *IsDiagonalOrientation*.

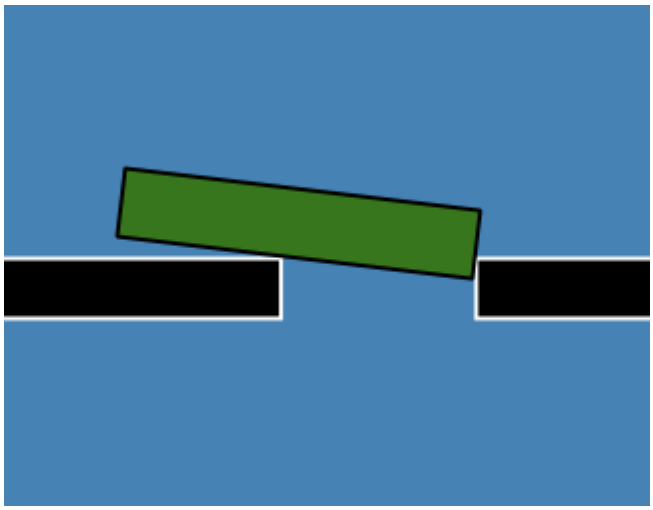

Fig. 5. Get stuck at a gap

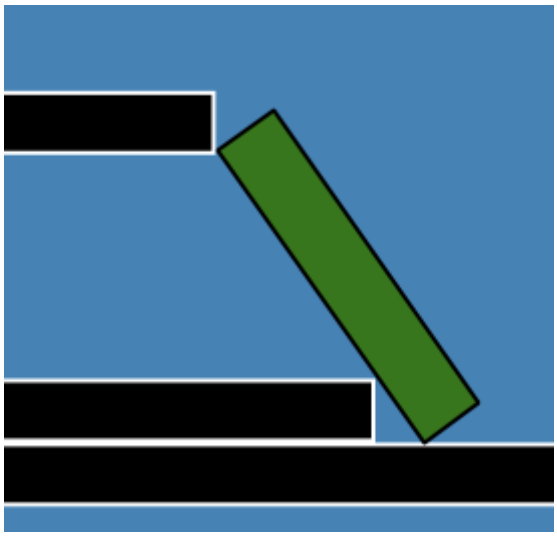

Fig. 6. Get stuck in a diagonal position

III. Conclusion

A brief overview of the Subgoal A* Agent was given. A fast domain dependent abstraction for the rectangle player of Geometry Friends has been proposed. The abstraction is based on a directed graph with nodes and edges. In Subgoal A* an extra property is added to the A* node to store the number and order of collected diamonds. This technique uses A* to search for the complete route with all diamonds or decreases the number of diamonds to find a route with fewer diamonds. A rule-based driver has been created to follow the calculated route of the search technique node by node. It returns the actions to perform. For more information please read the master thesis "DEVELOPMENT OF SEARCH-BASED AGENTS FOR THE PHYSICS-BASED SIMULATION GAME GEOMETRY FRIENDS".

IV. Files

- MasterThesisDanielFischerFinalMarch2015.pdf is my master thesis
- Fischer.xml are some levels I created for testing
- All other .cs files are for the Subgoal A* Agent, also MCTS.cs etc. because the agent's behaviour can easily be changed in the RectangleAgent.cs in line 85 *int runAlgorithm = 5;* 5 is for Subgoal A*. The same can be set in Driver2.cs in line 41 *private int runAlgorithm = 5;* 5 is also for Subgoal A*. I did not want to rewrite everything for the competition, so that all .cs files are necessary.

References

[1] Kim, H.-T., Yoon, D.-M., and Kim, K.-J. (2014). Solving Geometry Friends using Monte-Carlo tree search with directed graph representation. Computational Intelligence and Games (CIG), 2014 IEEE Conference on, pp. 462–463, IEEE. [3, 9, 10, 23]

[2] Perez, D., Powley, E., Whitehouse, D., Rohlfshagen, P., Samothrakis, S., I., Cowling P., and Lucas, S. (2013). Solving the Physical Traveling Salesman Problem: Tree search and macro-actions. IEEE Transactions on Computational Intelligence and AI in Games, Vol. 6, No. 1, pp. 31–45. [2, 3, 9]