International Core Journal of Engineering Volume 6 Issue 12, 2020
ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

Planning for Overcooked Game with PDDL

Yuxin Liu[1], Qiguang Chen[2], Ke Jin[3], Zhanhao Zhang[4]

[1]Faculty of Engineering, the Chinese University of Hong Kong, New Territories, Hong Kong (SAR), China;

[2]Faculty of Engineering, Dalhousie University, Nova Scotia, Canada, B3J 1Z1, Canada;

[3]Shanghai Shangde Experimental school, Shanghai, 201315, China;

[4]Faculty of Engineering, Shandong University, Jinan, 250100, China.

Abstract

A typical Overcooked planning problem that requires agent to process ingredients and deliver dishes composed of multiple ingredients is addressed in this paper by utilizing the powerful planning language, PDDL. Compared to the original game, the basic problem is simplified to be single-agent, deterministic and fully-observable, with temporal features preserved. A solver that consists of a PDDL encoding of the problem along with an automatic encoder that encodes an initial game environment into a planning problem is designed and tested. On base of that, an attempt on using the encoding to solve multi-agent problems was presented, where certain level of cooperative behavior was obtained but obvious deficiencies also appeared in the plans.

Keywords

Overcooked, Planning, PDDL, Temporal planning, Multi-agent planning.

1. Introduction

Domain planning, an essential part of artificial intelligence, is more and more popular in recent years. To research the usage of domain planning in video games, a solver for Overcooked is created. This project aims to examine the possibility for a PDDL-based solver to plan in this game instead of humans.

PDDL which can solve real-world problems, is promising in solving the Overcooked game problem. To allow data processing and interfacing with the game or simulations, an extra abstraction layer was designed to assist the core PDDL planner. For an easy start, the fundamental problem is simplified to be single-agent, deterministic, and fully-observable. Further, the possibility of adding multiple agents is considered. As a relatively unattended method for Overcooked game problem, this work is expected to provide reference for other researches of the interest. Note that, for simplicity and considering technical limitations, the solver was only tested on a set of simplified game rules and without interfacing with a real game or any simulators.

1.1 Literature

About how to achieve a multi-agent system, There are two valuable articles in related fields are found, one of them is A Multi-Agent Cooperation Method in Dynamic Environment which extends to multi-agent and tetrad on the basis of single agent and triple[1]. The other one is Too many cooks: Bayesian inference for coordinating multi-agent collaboration, which improves the collaboration ability between agents by Bayesian Delegation[2].

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

2. Methodology

The definitions of simplified set of rules that the solver will respect is listed in section 2.1. The overall architecture that links the game and the core PDDL planner will be illustrated in section 2.2. Three main components of the solver, namely, the encoder, the PDDL planner and the executor will be described in the next three sections.

2.1 Problem formulation

To convert the game *Overcooked* into a planning problem, convoluted details of the game has been removed and some rules has been relaxed.

Map:

There are 7 types of tiles on the map: (1) Floor tiles: where the agent can walk on, (2) Kitchen counters: places where mobile objects can be placed and where dishes can be assembled on empty plates, (3) Chop boards: places where ingredients can be chopped into slices, (4) Stoves: places where chopped meat can be cooked, (5) Sinks: places where used plate can be washed, (6) Storages: places where materials are stored, and (7) Conveyers: places where dishes can be delivered, and used plates are received.

Movements:

As shown in Figure 1, the game has a grid-base map structure and continuous moves are allowed on 8 directions (axial and diagonal). In simplified problem, only discrete axial movements are allowed. A movement of distance 1 on the grid takes 1 second.

Actions:

Agent can pick up one object at a time and drop it at any location other than on the floor. Agent can chop ingredients at chop boards and cook meat at the stoves. Agent can assemble a dish in an empty plate. Agent can deliver a dish and receive cash. Chopping, cooking, and washing cost the agent 5 seconds and other action cost 0.1 second.

Goals:

The goal of the game is to deliver several hamburgers of different types (hamburger type depends on its ingredients).

2.1.1 Architecture

The core planning technique applied in the planning is temporal planning using PDDL2.1. However, the environment information explained in the last chapter is still too rich for a PDDL planner to obtain a plan in reasonable time. Thus, an 'abstraction layer' is placed between the planner and the original game world. This layer consists of an encode that processes the game data into a PDDL problem which is compatible with the PDDL domain defined for the game. After a plan has been generated by the planner, an executor will handle the low-level details and execute the plan.

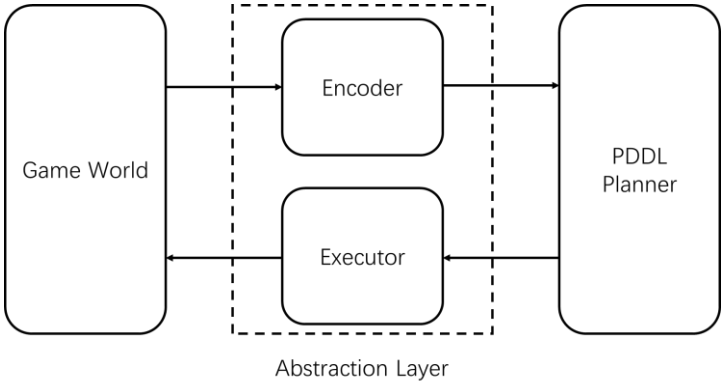


Figure 1: The architecture of the solver. The dashed frame encloses the abstraction layer that connects the game and the planner.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

2.1.2 Map Abstraction and Encoder

A topological graph is used in the problem abstraction for one major reason: retaining the topological feature of the original grid graph while at the same time, significantly shorten the length of each navigation sequence from one location to another. This will cut the depth of the search and very likely to improve the PDDL planner's searching time.

Basically, as shown in Figure 2, the topological map consists of two kinds of nodes, namely centers and functional locations. A center node represents the center of a local area of floor title. While functional location node represents a functional location like chop board or ingredient storage. Then two kinds of edges are presented in the graph, which are inter-area edges and intra-area edges. The former are edges connecting pairs of centers the areas of which are adjacent, and the latter are edges connecting functional locations and centers of areas from which they can be accessed directly. A functional location may be accessible from multiple areas, in this case the functional location has an edge to each of those centers. For ease of searching, kitchen counters are grouped together for each area and their location detail is ignored. Kitchen counters shared by multiple areas are represented preferentially to emphasize the possibility of objects passing over kitchen counter and for areas that does not share at all will be assigned a virtual kitchen counter whose distance to its center is set to be a specific uniform value, which in our test was set to 3.

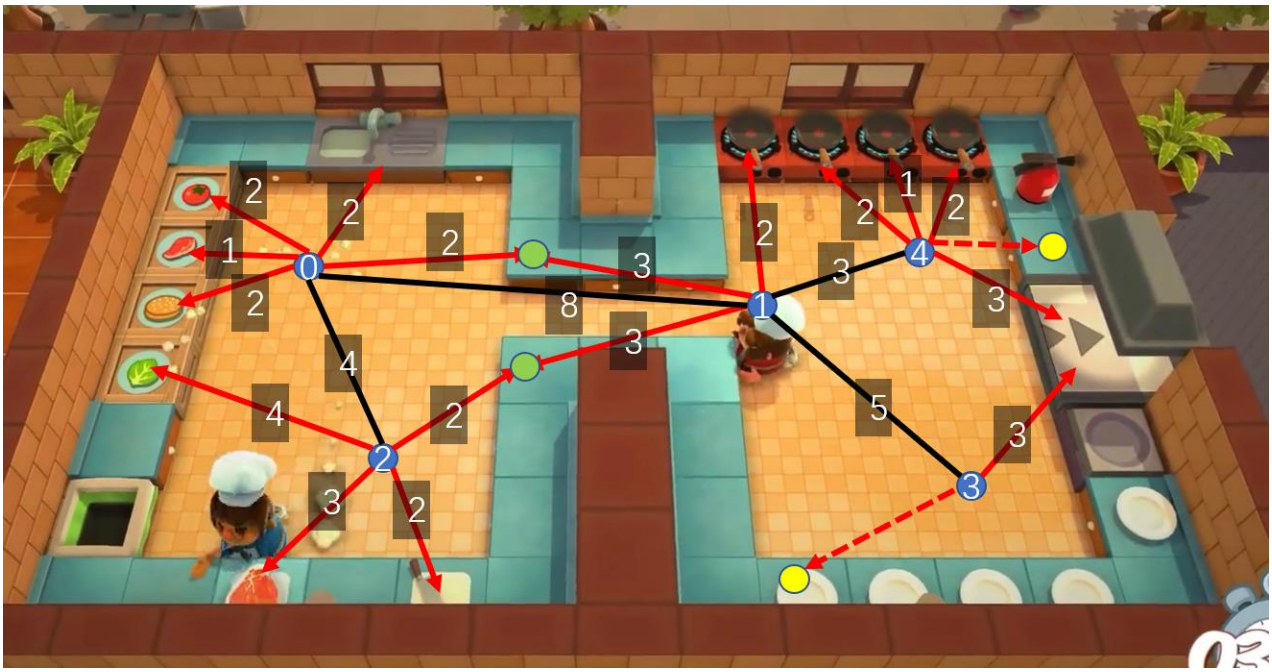


Figure 2: Example of topological representation of a game map. Blue nodes are area centers, green nodes are kitchen counters share by multiple areas and yellow nodes are virtual kitchen counters. Black edges are inter-area links, red edges are intra-area links and dash lines are virtual links.

For planning to operate automatically, a python program is used to perform the abstraction systematically. The K-mean algorithm is used to divide the floor areas into several local areas of the approximately close sizes [3]. The number of clusters was decided by specifying the average cluster size, which should neither be too large not too small to cover as much details as possible and at the same time consider the complexity. Note that because the performance of randomly placing the centers are likely to cause large size variance between clusters (areas), the idea of K-mean++ was borrowed to obtain finer initial center locations [4]. After the floor areas are clustered, connectiveness between each pair of nodes are tested, and in case two nodes have an edge between them, the distance between them are measured with Dijkstra algorithm on the original grid [5]. This information will then be encoded into a PDDL problem file the format of which will be introduced in the PDDL section.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

```
; KC  KC  SK  SK  KC  KC  KC  KC  ST  ST  ST  ST  KC
;IGT   0   0   0   0  KC  KC  KC   4   1   1   1  KC
;IGM   0   0   0   0  KC  KC  KC   4   1   1   1  KC
;IGB   0   0   0   0   4   4   4   4   4   1   1  CV
;IGC   2   2   2   2  KC  KC  KC   4   4   3   3  CV
; KC   2   2   2   2  KC  KC  KC   3   3   3   3  PR
; TC   2   2   2   2  KC  KC  KC   3   3   3   3  KC
; KC  KC  CB  KC  CB  KC  KC  KC  KC  KC  KC  KC  KC
;table0: (0, 4)
;table1: (2, 4)
;table2: (1,)
;table3: (3,)
```

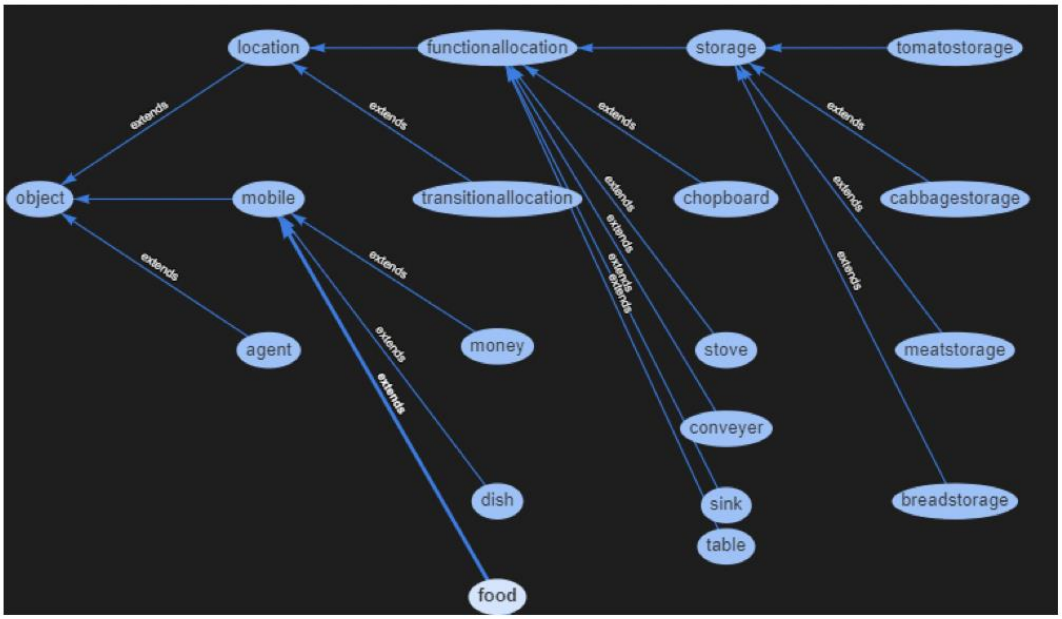
Figure 3: Example of an automatic floor area partition with K-means algorithm (floor tiles are denoted with its corresponding area id). Kitchen counters (named as tables) are generated according to the rules. Table0 and table1 are shared counters and table2 and table3 are virtual ones.

2.2 PDDL Planner

PDDL is intended to define the "physics" of a domain, that is, what predicates there are, what actions are possible, what the structure of compound actions is, and what the effects of actions are. Most planners require additional advice," that is, annotations about which actions to use in attaining which goals, or in carrying out which compound actions, under which circumstances." [6] As the core planning method used in this project, PDDL can analyze the initial and goal state and generate a plan to change the states of objects between them.

2.2.1 PDDL Domain

Though PDDL can generate plans for any task in *Overcooked* theoretically, the only problem to be solved in this project is the hamburger producing process. Hamburgers in *Overcooked* are composed of cabbage slice, tomato slice, roast steak, and bread (there are two variants, one with no tomato and one with no vegetable at all). This section will introduce the PDDL domain in three aspects: types, predicates, function, and action.



(a)

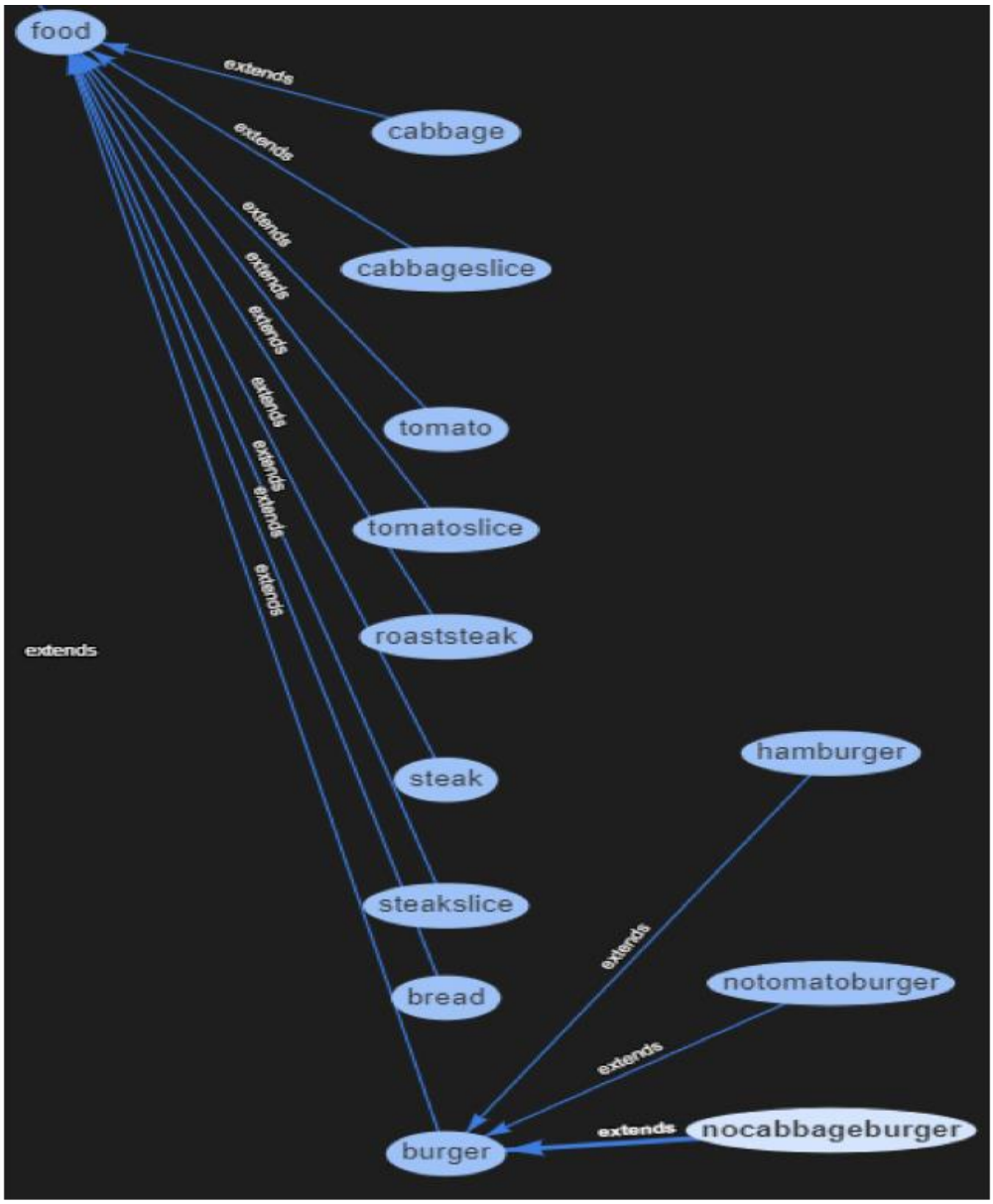
Figure 4: The hierarchy graph of (a) objects.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

To make sure that PDDL can produce a reasonable plan, the relation between objects is significant. A hierarchical object structure was defined with PDDL syntax to represent the relationships between objects, as shown in Figure 4. This is considered vital to make the planner respect the argument types of each action and at the same time allow some actions to be performed among several sibling subtypes. The objects in the PDDL domain are divided into three types (shown in Figure 4a): location, agent, and mobile.

There are six locations defined in the code, such as the stove and chop board, according to the composition of maps described in the problem formulation, except floor tiles. This guarantees that each specific action is done in a corresponding area. To reproduce the original map structure, each type of functional location will have its correspondence in the type definition and also object definitions for each of its instance in the problem file which will be described later.

Agent type is defined to represent and distinguish agents. This is important to record the state information of each individual agent in a multi-agent setting, about their location, the object they carry and whether they are busy.



(b)

Figure 4 cont.: The hierarchy graph of (b) food.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

Type mobile is defined as all objects which can be carried by agents, such as ingredient and dish. This type is defined to differ objects that can be picked up by agents from all others, to prevent agents or locations to be picked up. Its subtype mobile are money, dish, and food. Money can be obtained by agents when they sell hamburgers to customers. Dishes are used to contain hamburgers, and food (details are shown in Figure 4b), which is one of the most important types, includes all materials, such as tomato and cabbage, and cooked food roasted steak and hamburgers. To ensure that hamburgers of different ingredients are produced to meet customers' demand, three kinds of hamburgers are defined in the type hamburger to specify the goals.

As shown in Figure 5, several predicates are used to define the location and states of objects. The type specification of each argument is also defined. For example, (at ?b - object ?r -location) state the location of an object. The predicate (busy ?g- agent) represent the busyness of an agent and act as the precondition of every action, so agent can only work on one action at a time. The state (not(busy ?g - agent)) is added in the effect of each action, which means that the agent can do the next action when it finishes the action it was working on. (free ?g - agent) means that the agent does not carry any object, which is the precondition of action "pick" and the effect of action "drop". (carry ?b - mobile ?g - agent) is used to define that the agent carries a mobile object. This predicate allows the agent can move to a location with the object carried by it. The predicates (clean ?a- dish) and (in ?a - burger ?b - dish) are related to the process in which agent assemble hamburgers in a clean dish. As shown as system structure part, (path ?x ?y - location) is the precondition of action move. If a path does not exist between two locations, agents cannot move between them. The last predicate, (materialused ?a - mobile), is used to immobilize used material from being reused.

```
(:predicates
        (at ?b - object ?r - location)
        (free ?g - agent) 2👁 1☑ 3☒
        (busy ?b - agent) 14👁 14☑ 14☒
        (carry ?b - mobile ?g - agent) 1👁 3☑ 1☒
        (clean ?a - dish) 4👁 1☑ 7☒
        (path ?x ?y - location) 1👁
        (moving ?a - location ?b - location) 1☑ 1☒
        (in ?a - burger ?b - dish) 3👁 3☑ 6☒
        (materialused ?a - monile) 17👁 17☑
)
```

Figure 5: All predicates used in domain file.

The third part is functions, which are used to define a value in domain and problem file. As shown in Figure 6, the distance between two locations can be represented by a value [7].

The last part is action. The basic process to achieve the goal states of *Overcooked* must be described to introduce this part. These processes are: (1) Take out the materials from the storage where the materials are stored, (2) Chop materials to slices in the chop board, (3) Cook materials in the stove, (4) Put cooked food on the empty dish, (5) Assemble designated food combination (in this study, hamburger) on the table and sell it to customers at the conveyer, and (6) Send dirty dishes from the conveyer to sink and wash them.

```
(:functions
    (distance ?from - location ?to - location)
)
```

Figure 6: All functions used in domain file.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

The method which is used to define actions in the domain file are shown in Figure 7. Durative actions mean that actions need to finish in several seconds [8]. For example, agents need to spend 5 seconds to chop vegetables. Actions pick, move, and drop give agents the ability to change the location of movable objects. The duration time of action "move" is based on the distance between the two locations defined in the problem file.

A series of chop action must be finished at the location "chop board", and the effect of this action is converting material to material slice, for example, from cabbage to cabbage slice. The action cook is used to convert a steak slice to roast steak in location "stove". The action "assemble" is used to produce different kinds of hamburgers by compositing cabbage slice, tomato slice, roast steak, and bread. The action "wash" is used to convert not clean dishes to clean dishes. Finally, the action "sell" is used to sell hamburgers to customers and get money.

```
    (:durative-action pick ···
    )
    (:durative-action move ···
    )
    (:durative-action drop ···
    )
    (  :durative-action chopsteak ···
    )
    (:durative-action cook ···
    )
    (:durative-action choptomato ···
    )
    (:durative-action chopcabbage ···
    )
    (:durative-action assemble ···
    )
    (:durative-action wash ···
)
    (:durative-action sell ···
)
(:durative-action sellnotomato ···
)

(:durative-action assemblenotomato ···
(:durative-action sellnocabbage ···
)

  (:durative-action assemblenocabbage ···
)
```

Figure 7: All durative actions used in domain file.

According to the test in the experiment section, this design of encoding succeeded in planning on simple tasks but exceeded the time limit when facing more complicated tasks. For this reason, a simple modification was made to improve time efficiency, by combining actions with moves (this version of encoding will be denoted as combined encoding in the rest of the paper). The different result of encodings will be compared in the experiment section. Recall that there are area centers and functional locations introduced in the map abstraction section, in the combined encoding, action move will only be used to move between area centers instead of arbitrary pair of locations. And when the agent needs to perform a task at a functional location, for example chop board, it won't need to move to that location and back to the center; instead it can conduct the action right at the center, as long as

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

there exists an edge connecting the center and that functional location. In the case of chop action, the agent can start from the area center which has a chop board in the area with a unprocessed ingredient in its hand, perform action chop, it will take the time of chopping plus the round trip time between the center and the chop board, and when finished, it will be standing at the center like it was before the action, but with a chopped ingredient in its hand. It's straight forward to tell that a great many move actions which used to navigate the agent between centers and functional locations are eliminated and should also reduce the chance of taking meaningless moves.

2.2.2 PDDL Problem

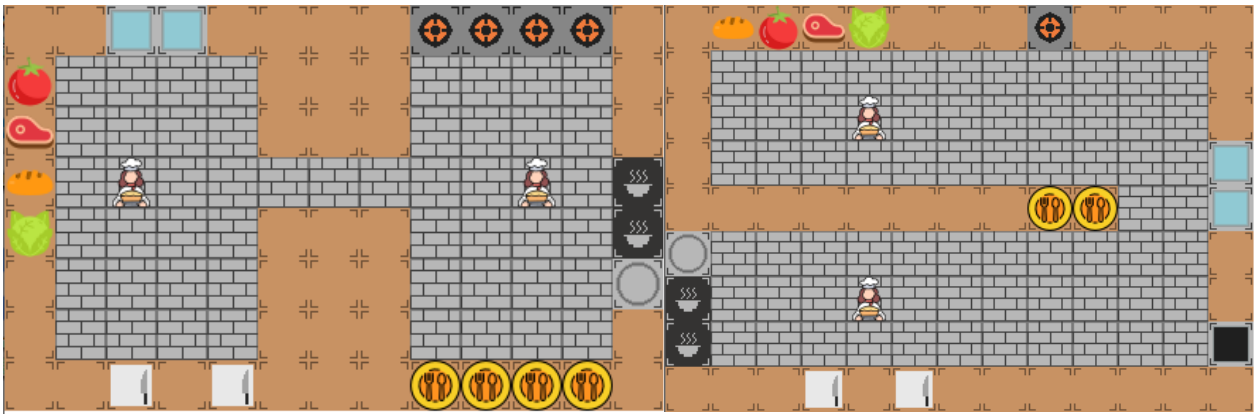
There are three main definitions in a PDDL problem definition: objects, initial state, and goal state. The objects part is used to define the most fundamental objects [9]. These objects are instances of the types. Initial state defines the initial predicates as well as numeric fluent, for example (at steak1 meatstorage) and (= (distance blue0 blue2) 4), including all initial information about objects and distance between locations. The goal state defines the final goals of plan, for example, (at money1 table1).

2.3 Executor

As mentioned above, an executor will be used to receive plan from the planner and then execute the plan by breaking it down into simple low-level control input that the game can receive, and possibly also deal with the conflicts in the plan on the execution level, for example when multiple agents are demanding access to a same location on their navigation routes, or when one plan has its precondition not met because of an unexpected overrun of another plan. Due to the time limitation of the project, the executor has been implemented yet.

3. Experiment and result

In this investigation, two maps are generated, simple map A and complex map B, which is shown in Figure 8a. and Figure 8b. The difference in difficulty between these two maps are determined by the average degree of the nodes in the topological map one will be converted to. In the map A, two main areas are almost isolated from each other and there are barely any shared locations between those area. While in the map B, two areas are crossed in the center by a long line of kitchen counters on which the objects can be put on. Due to the existence of this long kitchen counter, more edges connecting these shared locations would be generated in the topological map, more actions related to utilizing these shared locations will be available for the agents, and as a result, the branching factor of planning is increased.



a) b)

Figure 8: Illustration of (a) map A and (b) map B.

The Temporal Fast Downward planner is used to test the performance of two versions of PDDL encoding described in the PDDL section (original encoding and encoding with combined actions) for

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

making one hamburger in both types of maps. Because the original encoding failed on all complex problems, only the performance of the combined encoding was tested for different number of agents and tasks composed of two orders. During the comparison, several important indicators are focused on, including: (A) 'Success' which reflects whether the PDDL planner are able to complete a certain task within a certain time limit (set to be 1000s in these tests), (B) 'Searching time' which quantitatively measures the efficiency of the searching, (C) 'Duration' which indicates the expected execution time of the plan.

Table 1: Performance results for two types of methods for making a hamburger by single agent in simple map

	One agent One hamburger in simple map					
	Success	search time	Generated nodes	average branching factor	steps	Duration
Original Encoding	Yes	120s	146482	6.94	78	187.07
Encoding with Combined Actions	Yes	20s	24049	8.34	29	181.37

Table 2: Performance results for two types of methods for making a hamburger by single agent in complex map

	One agent One hamburger in complex map					
	Success	search time	Generated nodes	average branching factor	steps	Duration
Original Encoding	No	1000s+	1060526	7.8	None	None
Encoding with Combined Actions	Yes	50s	80778	8.31	37	187.66

Table 3: Performance results for different agents for making one hamburger in complex map by using advanced method

	One agent One hamburger in complex map					
	Success	search time	Generated nodes	average branching factor	steps	Duration
One agent	Yes	50s	80778	8.31	37	187.66
Two agents	Yes	20s	10546	11.04	44	125.1

Table 4: Performance results for different agents for making two hamburgers in complex map by using advanced method

	Produce two hamburgers in complexed map					
	Success	search time	Generated nodes	average branching factor	steps	Duration
One agent	No	1000s+	1000000+	7.67	None	None
Two agents	No	1000s+	1000000+	11.35	None	None

Since the combined encoding merges movements precede and succeed an action into that action, it would decrease the computation complexity significantly. From Table 1 and Table 2, the search time of the planner operating with the combined encoding is considerably shorter and makes it practical to plan on complex maps. In addition, according to the output in Figure 9, planner came up with a strategy of using the kitchen counter in the middle to reduce the number of round trips between storages and chop boards, showing that the generated abstract map can help planner utilize terrain features.

From Table 3, the search time was shortened when a second agent was introduced, which is relatively counter-intuitive. By looking into the log information of the planner, possible cause of this result is that the planner hit a goal state in a single round in multi-agent planning while reworked several times in the single-agent problem. Also notice that there was a 60-second decrease in duration time in multi-

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

agent plan compared to the single-agent one, which indicates that there should be some level of collaborations between two agents. Refer to Figure 10, the agent whose initial location is close to the storage naturally takes charge of transporting raw ingredients. However, a deficiency appeared that one agent idles for a long time, and sometimes repeat a couple of futile actions.

```
0.00100000: (move agent1 blue6 blue3) [6.00000000]
0.00100000: (pick agent0 cabbage0 cabbagestorage0 blue2) [6.10000000]
6.01100000: (pick agent1 dish0 table2 blue3) [6.10000000]
6.11100000: (drop agent0 cabbage0 table2 blue2) [4.10000000]
10.22100000: (pick agent0 cabbage0 table2 blue2) [4.10000000]
12.12100000: (move agent1 blue3 blue6) [6.00000000]
14.33100000: (move agent0 blue2 blue0) [4.00000000]
18.34100000: (drop agent0 cabbage0 table0 blue0) [4.10000000]
22.45100000: (pick agent0 bread0 breadstorage0 blue0) [6.10000000]
28.56200000: (drop agent0 bread0 table0 blue0) [4.10000000]
32.67200000: (pick agent0 tomato0 tomatostorage0 blue0) [4.10000000]
36.78200000: (drop agent0 tomato0 table0 blue0) [4.10000000]
40.89200000: (pick agent0 steak2 meatstorage0 blue0) [6.10000000]
47.00200000: (move agent0 blue0 blue2) [4.00000000]
18.13100000: (drop agent1 dish0 table0 blue6) [4.10000000]
51.01200000: (move agent0 blue2 blue4) [4.00000000]
22.45200000: (pick agent1 cabbage0 table0 blue6) [4.10000000]
26.56200000: (chopcabbage agent1 cabbage0 chopboard0 blue6 cabbageslice2) [11.00000000]
55.02200000: (move agent0 blue4 blue5) [4.00000000]
59.03200000: (move agent0 blue5 blue1) [4.00000000]
63.04200000: (move agent0 blue1 blue3) [4.00000000]
37.57200000: (drop agent1 cabbageslice2 table0 blue6) [4.10000000]
67.05200000: (chopsteak agent0 steak2 chopboard1 blue3 steakslice0) [13.00000000]
```

Figure 10: Part of a plan in which one agent processed the ingredient transported by the other agent. Also, there are several useless moves mainly taken by agent0 including picking-dropping loop.

```
0.00100000: (move agent0 blue1 blue5) [4.00000000]
4.01100000: (pick agent0 dish0 table3 blue5) [4.10000000]
8.12100000: (move agent0 blue5 blue1) [4.00000000]
12.13100000: (move agent0 blue1 blue3) [4.00000000]
16.14100000: (drop agent0 dish0 table0 blue3) [4.10000000]
20.25100000: (pick agent0 bread1 breadstorage0 blue3) [6.10000000]
26.36100000: (drop agent0 bread1 table0 blue3) [4.10000000]
30.47100000: (pick agent0 steak0 meatstorage0 blue3) [6.10000000]
36.58100000: (move agent0 blue3 blue1) [4.00000000]
40.59100000: (drop agent0 steak0 table1 blue1) [4.10000000]
44.70100000: (pick agent0 cabbage0 cabbagestorage0 blue1) [6.10000000]
50.81100000: (move agent0 blue1 blue5) [4.00000000]
54.82100000: (move agent0 blue5 blue2) [4.00000000]
58.83100000: (move agent0 blue2 blue0) [5.00000000]
63.84100000: (move agent0 blue0 blue6) [5.00000000]
68.85100000: (chopcabbage agent0 cabbage0 chopboard1 blue6 cabbageslice1) [9.00000000]
```

Figure 9: Part of a plan in which the agent showed the ability to stack ingredients on kitchen counters (tables) before moving a long way to the opposite side of the table where ingredient storages are not accessible.

Finally, from Table 4, the time complexity is unaffordable for multiple order tasks, there is obvious difficulty in planning to meet lengthy goals.

4. Discussion

While the PDDL domain achieves the initial goal that multiple agents can work and collaborate, there are still some limitations that can be further investigated. The biggest potential problem is that the

ISSN: 2414-1895 DOI: 10.6919/ICJE.202012_6(12).0047

solver still cannot achieve close optimal efficiency although the time consumed by multiple agents is always shorter than that of single agents. As there were not any obvious fault in the encoding that could be spotted, it is suspected that the heuristic of the planner affects the task distribution.

Another weakness is that the search time used to output the result lasts too long, the planner struggles to search for a plan especially for complex maps or multiple orders. Obviously, it is possible to separate orders so that they can be dealt with one by one though, it is still worth working on to see if any method could produce several hamburgers at the same time without hurting the performance conspicuously. One possible way might be breaking down the tasks outside the PDDL into categorical goals that releases the pressure on the searching.

5. Conclusion

In this project, a PDDL domain is successfully developed and improved to solve a typical *Overcooked* planning problem in context of multiple agents cooking a meal together. This PDDL domain theoretically allows agents to collaborate and work in parallel, but only relatively low-level collaborations are achieved by searching. PDDL planner, encoder and executor will work together to finish the task. The difficulty of planning on a complex task showed that design effort had to be made in order to plan with PDDL practically in a game. In the future, the design attempts will be made both within and outside PDDL to support more complex problem. Besides more objects, new workspace and new food, will also be added to increase program's adaptability when it meets different demand.

References

[1] Wu K., Liu W., Li S., Wang J.(2017).A Multi-Agent Cooperation Method in Dynamic Environment Vol.39(2), pp.186.

[2] Wang, R., Wu, S., Evans, J., Tenenbaum, J., Parkes, D., & Kleiman-Weiner, M. (2020). Too many cooks: Bayesian inference for coordinating multi-agent collaboration. pp. 1.

[3] Lloyd, S. (1982) Least squares quantization in PCM. IEEE transactions on information theory, 28(2): pp. 129-137.

[4] Arthur, D., Vassilvitskii, S. (2007) k-means++: The advantages of careful seeding. In: Symposium on Discrete Algorithms. USA. Pp. 1027-1035.

[5] Dijkstra, E. W. (1959) A note on two problems in connexion with graphs. Numerische Mathematik, 1(1): pp. 269–271.

[6] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., & Weld, D. (1998). PDDL -The Planning Domain Definition Language. pp. 1-5.

[7] Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. JInal of Artificial Intelligence Research, 20, 61–124. https://doi.org/10.1613/jair.1129. pp. 5-11.

[8] Fox, M., & Long, D. (2003). PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. Journal of Artificial Intelligence Research, 20, 61–124. https://doi.org/10.1613/jair.1129. pp. 1-25.

[9] Ghallab, M., Knoblock, C., Wilkins, D., Barrett, A., Christianson, D., Friedman, M., Kwok, C., Golden, K., Penberthy, S., Smith, D., Sun, Y., & Weld, D. (1998). PDDL -The Planning Domain Definition Language. pp. 18-19.