# Agent Decision Making and Planning

William John Lautama

# Planning Model

## World Variables

For the world variables, I used the world variables that are already present in the world. The main variables that are present are TotalWood, TotalStone, TotalGrain, AgentWood, AgentStone, AgentGrain, and NumBuildings. Here are a brief description of what each variable represents and why I chose the variable to work in the planning model :

- **TotalWood** : This variable keeps track of the total wood that is present in the world. In order for me to make the ships have a goal of collecting wood, I have to adjust the variable TotalWood.
- **TotalStone** : This variable keeps track of the total stone that is present in the world. In order for me to make the ships have a goal of collecting stone, I have to adjust the variable TotalStone.
- **TotalGrain** : This variable keeps track of the total grain that is present in the world. In order for me to make the ships have a goal of collecting grain, I have to adjust the variable TotalGrain.
- **AgentWood** : Each agent has an inventory that they are carrying. In order for me to adjust the TotalWood correctly and check what kind of resource the agent has got, I have to use AgentWood. This can help with keeping track of the plan, especially if the goal revolves around increasing the TotalWood.
- **AgentStone** : Each agent has an inventory that they are carrying. In order for me to adjust the TotalStone correctly and check what kind of resource the agent has got, I have to use AgentStone. This can help with keeping track of the plan, especially if the goal revolves around increasing the TotalStone.
- **AgentGrain** : Each agent has an inventory that they are carrying. In order for me to adjust the TotalGrain correctly and check what kind of resource the agent has got, I have to use AgentGrain. This can help with keeping track of the plan, especially if the goal revolves around increasing the TotalGrain.
- **NumBuildings** : In order for me to make a goal for the builders to keep building, I have to keep track of the amount of buildings there are in the world and then keep on building more from it. Therefore, I used the NumBuildings variable to keep track and to make the goal for the builder when the builder needs to build.

## Actions

There are 3 actions in total that I implemented in my GOAP planner. It is also worth mentioning before that I am using **Forward Planning** for the GOAP planner so the preconditions and the effects are all based on the forward planning method. The justification for the plan will be explained in the **Planning Approach** section. Here are the actions with each of their own explanations and justifications for their Preconditions and Effects :

## CollectResources

### Preconditions

For the preconditions, in order for me to collect the resources, I need to have the ship have 0 resources in the inventory. This is a very simple concept therefore, I used the **AgentWood**, **AgentStone**, and **AgentGrain** variables and made sure that the variables are set to 0. This is because due to the rules, the ship isn't allowed to collect resources when it has a resource. Here is a pseudocode of the implementation :

```
If AgentWood is 0 and if AgentStone is 0 and if AgentGrain is 0

      Precondition is true

Precondition is false
```

With this implementation, the precondition of the collected resources should work accordingly with the GOAP planner.

### Effects

For the effects, I used a very specific strategy for the GOAP planner. I noticed that building universities is a very dominant strategy therefore, I made sure that the ships collect the resources that are needed to make universities. But, it's not as simple as for every ship, collecting resources needed. It's more complex than that. Since each ship has their own role and resources that they can get more efficiently than other ships (except the builder), each ship will be focusing first on their own resources, then each ship will help the other resources that are not in their specialty.

For example, WoodCutter ships checks if wood is less than 15 first since it is how much needed to make a university. If it is more than 15, then it checks if stone is less than 10, and if stone is more than 10 it checks if grain is less than 5.

This goes on with the stonemason and farmer but with different order with stonecutter going
With Stonemason going :
Stone -> Wood -> Grain

And Farmer going :
Grain ->Wood -> Stone

And lastly Builder going :
Wood ->Stone -> Grain

This order is not picked randomly but it is set like that so that it checks the resources with the least amount of time to collect. After checking all of the resources and if all of the resources met the criteria of building the university, there is another layer of checks that I added. I noticed that there are only 3 builders in the world therefore the maximum amount of resources needed to make universities are only 3 universities at once. So, the total

resources should only have the needed 3 times the amount needed to make the university. So I did the exact same checks as I did previously but I tripled the amount which looks like :

Woodcutter :
Wood (Check if less than 45) -> Stone (Check if less than 30) -> Grain (Check if less than 15)

Stonemason : Stone ->
Wood -> Grain

Farmer :
Grain -> Wood -> Stone

Builder :
Wood -> Stone -> Grain

*Note that each role checks the same amount of resources as the woodcutter which is 45 wood, 30 stone, and 15 grain.

If after all of the checks, everything is more than what is needed, each ship will collect the resources that they are the most efficient with except for the builder meaning,

Woodcutter -> Collect Wood (AgentWood increases)
Stonemason -> Collect Stone (AgentStone increases)
Farmer -> Collect Grain (AgentGrain increases)
Builder -> Collect Wood (AgentWood increases)

The reason why the builders are collecting wood is because wood is the resource that needs the shortest amount of time to collect and since I want the builder to focus on building universities, I want it to collect resources in as little time as possible. Here is the pseudocode of the implementation :

```
If AgentType is Woodcutter
      If TotalWood < 15
            Collect Wood (Increase AgentWood)
      Else If TotalStone < 10
            Collect Stone (Increase AgentStone)
            Else If TotalGrain < 5
            Collect Grain (Increase AgentGrain)
      Else
            If TotalWood < 45
                  Collect Wood (Increase AgentWood)
```

```
            If TotalStone < 30
                  Collect Stone (Increase AgentStone)
            If TotalGrain < 15
                  Collect Grain (Increase AgentGrain)
            Else
                  Collect Wood (Increase AgentWood)
Else if AgentType is Stonemason
      If TotalStone < 10
            Collect Stone (Increase AgentStone)
      Else If TotalWood < 15
            Collect Wood (Increase AgentWood)
      Else If TotalGrain < 5
            Collect Grain (Increase AgentGrain)
      Else
            If TotalStone < 30
                  Collect Stone (Increase AgentStone)
            If TotalWood < 45
                  Collect Wood (Increase AgentWood)
            If TotalGrain < 15
                  Collect Grain (Increase AgentGrain)
            Else
                  Collect Stone (Increase AgentStone)
Else if AgentType is Farmer
      If TotalGrain < 5
            Collect Grain (Increase AgentGrain)
      Else If TotalWood < 15
            Collect Wood (Increase AgentWood)
      Else If TotalStone < 10
            Collect Stone (Increase AgentStone)
      Else
            If TotalGrain < 15
                  Collect Grain (Increase AgentGrain)
            Else If TotalWood < 45
                  Collect Wood (Increase AgentWood)
            Else If TotalStone < 30
                  Collect Stone (Increase AgentStone)
            Else
                  Collect Grain (Increase AgentGrain)
```

It is also worth mentioning that when increasing the AgentWood, AgentStone, or AgentGrain, the resource that the ship is going to needs to be checked if it has only 1 left and the ship is the ship that usually collects 2 at once, it needs to only collect one regardless for the lack of the resource on the resource node. With this implementation, this should help the ship collect the resources as efficiently as possible to generate more points in the simulation.

## DepositResources

### Preconditions

The way the precondition works for the DepositResources is very simple. It needs to check if the ship has any type of resources in the inventory. If it does, it returns the precondition as true. This is because based on the rules, the ship needs to immediately deposit the resource after collecting it and this can be done by checking if AgentWood, AgentStone or AgentGrain is more than 0. Here is the pseudocode of the implementation :

```
If AgentWood > 0 or AgentStone > 0 or AgentGrain > 0
      Return true
Return false
```

By using this precondition, this can ensure that the ships deposit the resources as soon as they collect it properly.

### Effects

The effects of the DepositResources action is very simple. It basically adds up the TotalWood, TotalStone and TotalGrain variables with the AgentWood, AgentStone, and AgentGrain variables. This is done because each ship can only carry 1 type of resources each time so if each resource is added, it still won't affect the uncollected resources since it will just increase it by 0. Then, the AgentWood, AgentStone, and AgentGrain needs to be set to 0 to simulate the agent emptying the inventory. Here is the pseudocode of the implementation :

```
TotalWood += AgentWood
AgentWood = 0
TotalStone += AgentStone
AgentStone = 0
TotalGrain += AgentGrain
AgentGrain = 0
```

By implementing these simple effects, this will simulate the ships emptying and depositing the resources into the world properly through the GOAP planner.

## BuildAction

### Preconditions

The precondition for the BuildAction is also very simple. It checks to see if the ship has 0 resources all around (AgentWood, AgentStone, AgentGrain) since builders are not allowed to build if they have 0 resources. Then, it also checks to see if the resources are enough to build the building needed which is the university since the university gives the most points. Here is the pseudocode of the implementation :

```
If (AgentWood is 0 and AgentStone is 0 and AgentGrain is 0)
      If (TotalWood >= 15 and TotalStone >= 10 and TotalGrain >= 5)
            Return true
Return false
```

This implementation of the precondition will make for a proper and working build action for the ships.

### Effects

The effects of the BuildAction are also very simple. It adjusts the value of the TotalWood, TotalStone, and TotalGrain by the amount that is needed to make a university. Then, it adds the NumBuildings variable by 1 to keep track of the building count to set it up for the goal.

This is because the goal that I want to implement is to make more universities that are currently present therefore, keeping track of the NumBuildings variable is needed. Here is the pseudocode of the implementation :

```
TotalWood -= 15
TotalStone -= 10
TotalGrain -= 5

NumBuildings += 1
```

Although this is very simple, it is highly effective when deciding whether the builder is going to build or not.

## Goals

Each agent type has their own set of goals that they follow. I made 3 separate functions which are called GetWoodGoal, GetStoneGoal, and GetGrainGoal. Each of these functions serves its own purpose which are :
GetWoodGoal : Get more wood than the world currently have
GetStoneGoal : Get more stone than the world currently have
GetGrainGoal : Get more grain than the world currently have

These functions then go into the same checks as the one on collecting resources with it checking the amount of resources needed based on the checks. This means that each ship’s priority is based on their type with Woodcutters focusing on wood, Stonemasons focusing on stone, and Farmers focusing on grain but being able to collect other resources based on what resources are needed based on the checks. Lastly, the Builder’s main priority is building more buildings than the current number of buildings in the world. But if the builder is not able to build, the builder's priority will be collecting the resources the same way a woodcutter would do since wood takes the shortest amount of time to collect. Here is the pseudocode of the implementation :

```
If AgentType is WoodCutter
      If TotalWood < 15
            GetWoodGoal
      Else If TotalStone < 10
            GetStoneGoal
      Else If TotalGrain < 5
            GetGrainGoal
      Else
            If TotalWood < 45
                  GetWoodGoal
            Else If TotalStone < 30
                  GetStoneGoal
            Else If TotalGrain < 15
                  GetGrainGoal
            Else
                  GetWoodGoal
Else if AgentType is StoneMason
      If TotalStone < 10
            GetStoneGoal
      Else If TotalWood < 15
            GetWoodGoal
      Else If TotalGrain < 5
            GetGrainGoal
      Else
            If TotalStone < 30
                  GetStoneGoal
            Else If TotalWood < 45
                  GetWoodGoal
            Else If TotalGrain < 15
            GetGrainGoal Else
                  GetStoneGoal
Else if AgentType is Farmer
      If TotalGrain < 5
            GetGrainGoal
      Else If TotalWood < 15
            GetWoodGoal
      Else If TotalStone < 10
      GetStoneGoal Else
            If TotalGrain < 15
                  GetGrainGoal
```

```
            Else If TotalWood < 45
GetWoodGoal
Else If TotalStone < 30
GetStoneGoal
Else
GetGrainGoal
Else
If TotalWood < 15
GetWoodGoal
Else If TotalStone < 10
GetStoneGoal
Else If TotalGrain < 5
GetGrainGoal
Else
If TotalWood < 45
GetWoodGoal
Else If TotalStone < 30
GetStoneGoal
Else If TotalGrain < 15
GetGrainGoal
Else
GetWoodGoal
```

This implementation will allocate the goal properly so that the ships can get the proper goal for the GOAP planner to search and get an optimal plan to reach the goal.

# Extras

## Scatter Method

To collect resources, I set it up so that the initial step of the ship is to scatter towards the four corners of the world. This only runs once at the start and never runs again. This is done to scatter the ship's location and for each ship to go to the resource location in their own region effectively. The effect of this is not major but it does affect the simulation in getting points in a positive way. Here is the pseudocode of the implementation :

```
Int InitialSetupCount++;
```

```
GridNode* UpperRight = Initialise upper right coordinates
GridNode* UpperLeft = Initialise upper left coordinates
GridNode* BottomRight = Initialise bottom right coordinates
GridNode* BottomLeft - Initialise bottom left coordinates
Float ShortestPath = 99999;

AActor* ClosestResource = nullptr

TArray of resources as actors initialised;

Get all resource actors and store it into TArray of resources

For (Resources TArray)
      If Resource is not valid
            Continue;
      If (Resource == ResourceType)
            Get all ship actors

            For (Ship Array)
                  If Resource's location == Ship's location
                        TempResource = Resource
      CurrentPath = distance of ship and resource
      If (CurrentPath < ShortestPath and Resource is not
TempResource)
            If (InitialSetupCount % 3 == 0)
            {
                  FirstAgent = Woodcutter
                  SecondAgent = Stonemason
                  ThirdAgent = Farmer
                  FourthAgent = Builder
            }
            Else If (InitialSetupCount % 3 == 1)
            {
                  FirstAgent = Stonemason
                  SecondAgent = Farmer
                  ThirdAgent = Builder
                  FourthAgent = Woodcutter
            }
            Else if (InitialSetupCount % 3 == 2)
            {
                  FirstAgent = Farmer
                  SecondAgent = Builder
                  ThirdAgent = Woodcutter
                  FourthAgent = Stonemason
```

```
        }
        Else if (InitialSetupCount % 3 == 3)
        {
              FirstAgent = Builder
              SecondAgent = Woodcutter
              ThirdAgent = Stonemason
              FourthAgent = Farmer
        }
        AShip* ShipClass = Cast ship
        If ShipClass's agent type == FirstAgent if(Resource
              is in upper right)
                    ShortestPath = CurrentPath
                    ClosestResource = Resource
        Else If ShipClass's agent type == SecondAgent
              if(Resource is in upper left)
                    ShortestPath = CurrentPath
                    ClosestResource = Resource
        Else If ShipClass's agent type == ThirdAgent
              if(Resource is in Bottom right)
                    ShortestPath = CurrentPath
                    ClosestResource = Resource
        Else If ShipClass's agent type == FourthAgent
              if(Resource is in bottom left)
                    ShortestPath = CurrentPath
                    ClosestResource = Resource

Return ClosestResource
```

Through this implementation, I will be able to scatter the ship efficiently at the start of the simulation for more effective resource collection.

## Trials & Experiments

Throughout the development, I experimented with different methods of the planning mode. At first, the planning model was to make the ships get their own respective resources that they are efficient at and make the builder get wood and build universities. The result of this implementation is approx. 15.000 - 17.000 points. This is not a bad result but I noticed the overflowing unused resources so I decided to optimise it.

The next method that I used is checking each of the ship's own resources matching their agent type and then helping the others to at least make 1 university. This is a more optimised result and the result from it is approx. 22.000 - 24.000 points. This is also not a bad result but I noticed that there are still overflowing resources so I optimised it more using the current planning model that I am using and the result of it is approx. 25.000 - 27.000 points which is a way better result compared to the results that were present before.

# Planning Approach

## Expander

For the expander, I used a forward search approach, where the expander function generates successors by working forward from the current state. In theory, backwards search is more efficient, but forward search is easier to understand conceptually in terms of preconditions and effects. The reason for this is because backwards search expands less nodes and forward nodes expand unnecessary nodes to get to the goal.

Due to the time constraints, I wasn't able to implement backwards search but I was able to implement forwards search properly into the simulation. There is a function called Expand who's job is to get the neighbour nodes that are currently present in the planner. Here is the pseudocode of the implementation of the expand function :

```
TArray GOAPNode* ConnectedNodes
TArray UHLAction AllowedActions

For (AvailableActions)
      UHLAction* CurrentAction = Make UHL action object of Ship
      If (CheckPrecondition)
            Add CurrentAction to AllowedAction

For (AllowedActions)
      GOAPNode* NextNode = new GOAPNode()
      NextNode's State = Node's State
      If (SetupAction)
            AllowedAction->ApplyEffects(Ship, NextNode->State)
            NextNode->Action = AllowedAction
            NextNode->RunningCost = 0
            NextNode->Parent = Node
            ConnectedNodes.Add(NextNode)

Return ConnectedNodes
```

The way this pseudocode works is that it goes through all the available actions. Then, it checks the precondition of the available actions and if it returns true, add it into the actions that the ships are allowed to do. After that, it goes through all the AllowedActions and changes the NextNode's state into the current Node's state. It then checks if the SetupAction goes through correctly and applies the effects of it into the ship and then adds the

AllowedAction as the next action with the Running Cost of 0 and sets the parent of the NextNode to the current Node. Lastly, it adds NextNodes to the connectedNodes and returns the ConnectedNodes array overall.

## Search

The way the planner works to find a solution is similar to how A* works for pathfinding but instead of pathfinding it works on states. I used forward planning so it starts from the start state but knows what the goal is. It uses the cost of each action which for the collect I set to 2 but for the rest I set to 1. Here is the pseudocode for the search planner :

```
Empty Ship's planned actions

GOAPNode* GoalNode = new GOAPNode();
GoalConditions = PickedGoal
GoalState

For (Goal conditions)
      Add to GoalState

GoalNode's state = GoalState
GoalNode's action = null
GoalNode's running cost = 0
GoalNode's parent = null

StartNode = new GOAPNode
StartNode's state = WorldState
StartNode's action = null
StartNode's running cost = 0
StartNode's parent = null

Initialise open and closed list

Open.Push(StartNode)

Int MaxRunningCost = 10

While (Open.Num() > 0)
      Float SmallestF = gets the F value to the goal state
      Int SmallestFIndex = 0

      For (Open list)
            Int CurrentF = get current F value for current state
            If (currentF < SmallestF)
```

```
                SmallestF = CurrentF
                SmallestFIndex = i

if(CurrentNode’s runningcost > MaxRunningCost)
      Return false

if(current state meets the goal) TArray<UHLAction*>
      ActionsToTake while(CurrentNode’s parent)
            ActionsToTake.Add(CurrentNode’s action)
            CurrentNode = CurrentNode’s parent

      For (ActionsToTake.Num())
            Add ActionsToTake to Ship’s PlannedActions

      Return true

TArray<GOAPNode*> ConnectedNodes = Expand(CurrentNode, Ship)

For (ConnectedNodes.Num())
      Int OpenTempTracker = 0
      Int ClosedTempTracker = 0

      For (Openlist) if(it's not the same state as the one in the
            open list
and connectednodes)
                OpenTempTracker++
      For (ClosedList) if(it's not the same state as the one in the
            closed list
and connectednodes)
                ClosedTempTracker++
      If (ClosedTempTracker == Closed.Num())
            Int PossibleG = CurrentNode’s running cost + Connected
Node’s action cost
            Bool bPossibleGBetter = false
      if(OpenTempTracker == Open.Num())
      Add ConnectedNode to open list
      bPossibleGBetter = true
      Else if (PossibleG < ConnectedNode’s running cost)
            bPossibleGBetter = true
      If (bPossibleGBetter)
            ConnectedNode’s parent = CurrentNode
            ConnectedNode’s running cost = PossibleG
Return false
```

Basically, the way it works is that it empties out the ship's planned actions. Then, it initialises the goal node based on the goal conditions. It also initialises the start node depending that will be connected to the goal node. After that it initialises the closed and open list that is going to be used in the search algorithm. It also initialises the max running cost at 10 so that the plan's maximum cost is 10. First, it gets the F value of the goal state and then the smallest F index. Then, it loops through an open list to set the smallest F value. It also takes into account the running cost with the maximum cost to make sure that the cost doesn't exceed the maximum cost.

Then, if the current state meets the goal state, It parents the nodes so that it can have a sequence of action leading to the goal state. Then, It adds the actions in the states that are parented into the array of the ship's planned actions.

It also goes through the neighbour nodes of the states and then decides which nodes are the best to the goal by tracking the open and closed list. This is done to figure out what the best G value is of all the actions that are connected to the current nodes. Then, it changes the G value based on what is calculated in the statements of the trackers of the open and closed list.

# Execution Considerations

## Plan Failure

In case a high-level action fails to proceed as expected (e.g. resource location is depleted, or an agent is unable to reach a resource slot), the planner will detect this failure and replan from the current world state. The replanning process will generate a new sequence of actions that takes into account the updated world state and any changes in resource availability or agent location. To handle plan failures more effectively, I implemented a simple boolean variable that cancels the plan when the target needs to be changed. This boolean is changed when the ship needs to change resources because the resource is either depleted or occupied. I also implemented extra checks of the new target that needs to be found if the resource is occupied by another ship that is called in the Replan function. Here is the pseudocode for the ship replan that is in the Replan function :

```
if(BlockedNode is a resource node) if(BlockedResources
        array is 0)
                Add BlockedNode's resource to array
        for(BlockedResources) if(BlockedNode's resource is in
        BlockedResources array) bBlockedResourceCheck = false Break
                Else bBlockedResourceCheck = true

                if(bBlockedResourceCheck)
                        Add BlockedNode's Resource to BlockedResources array
                        bBlockedResourceCheck = false

                If (Ship has PlannedActions)
                        Empty Path
```

```
        if(Ship's target exist)
                TempTarget = Target
                Target = Null
                Target = CalculateNearestGoal()
                Cancel execute
                Cancel setup
                SetupAction()
    Break
```

## Navigation Failure

To handle navigation failure, I used a replan function to avoid collision with other ships. Since I want the ships to not collide with each other as much as it can, it needs to replan the pathfinding algorithm but with an adjustment. The adjustment being re-running A* but with a blocked node that the algorithm cannot go through / has to skip through. The other thing to mention is the algorithm should also change targets if the resource is occupied with another ship. I made it a very simple algorithm that checks if the blocked node is a resource node. Then, it changes the ship's target into another resource with the same resource type. This is triggered every time the ship looks 1 step ahead and it collides with the other ship on that 1 step ahead that was looked into.

Although this is not the best solution, due to the time constraint, I decided to implement this solution to the simulation. Here is the pseudocode of the implementation in the Replan function for navigation :

```
Priority_queue OpenQueue;

Clear Ship's path;

ResetAllNodes();

GridNode* CurrentNode = Get current ship's grid location
G = 0;
H = CalculateManhattanDistance between goal and start
F = G + H
CurrentNode set is in open queue OpenQueue.Push(CurrentNode)

while(OpenQueue.size() > 0)
        CurrentNode = OpenQueue.top();
        OpenQueue.pop();
        CurrentNode set is not in open queue

        if(CurrentNode == Resource)
                Goal Found
                Break

        TArray<GridNode*> Neighbours = Get neighbours node

        For (Neighbours)
                GridNode* CurrentNeighbour = Neighbours[j];
                If (CurrentNeighbour is in closed)
                        Continue;
                If (CurrentNeighbour == BlockedNode)
                        Replan if BlockedNode is a resource that is occupied Continue
        Bool bPossibleGIsBetter = false
        Int PossibleG = CurrentNode's G + CurrentNeighbour's travel cost
        if(CurrentNeighbour is not in open) bPossibleGIsBetter = true
```

```
Else if(PossibleG < CurrentNeighbour’s G)
        bPossibleGIsBetter = true

if(bPossibleGIsBetter)
        CurrentNeighbour’s parent = CurrentNode
        CurrentNeighbour’s G = PossibleG
        CurrentNeighbour’s H = CalculateManhattanDistance between goal and Node
        CurrentNeighbour’s F = CurrentNeighbour’s G + CurrentNeighbour’s H
        CurrentNeighbour is set to in open queue
        OpenQueue.push(CurrentNeighbour)
```

This pseudocode represents the collision avoidance of when the ship is about to collide with another ship. Parts of the code also correlates with replanning the target when the target is occupied with a ship.

# Research Sources


Park, J., Lu, F.-C., & Hedgcock, W. M. (2017). Relative Effects of Forward and Backward Planning on Goal Pursuit. *Psychological Science*, *28*(11), 1620–1630. https://doi.org/10.1177/0956797617715510

*Goal Oriented Action Planning for a Smarter AI | Envato Tuts+*. (2014, April 23). Game Development Envato Tuts+. https://gamedevelopment.tutsplus.com/goal-oriented-action-planning-for-a-smarter-ai--cms-20793t

Orkin, J., & Productions, M. (n.d.). *Applying Goal-Oriented Action Planning to Games Applying Goal-Oriented Action Planning to Games*. from https://citeseerx.ist.psu.edu/document?doi=0c35d00a015c93bac68475e8e1283b02701ff46b&repid=rep1&type=pdf

*Total AI*. (n.d.). Totalai.org. http://totalai.org/doc-goap.html

Bjarnolf, P. & Institutionen för kommunikation och information. (2008). Threat analysis using Goal-Oriented action Planning. In *Examensarbete I Datalogi 30hp* [Thesis]. https://www.diva-portal.org/smash/get/diva2:2228/FULLTEXT01.pdf