Voices from Community

宠花上天

正在休产假中，社畜9年了，今年喜提儿子一枚，第一次那么长的假期，趁着产假看看学学，本来觉得和自己工作业务毫不相干，意外找到了很多相似的地方，比如引擎里的消息机制，其实自动驾驶的底层也是类似这么玩儿的--ros消息订阅发布，有点意思哈哈哈。写了不少字了，，，因为所以，，，，课程小秘书，，，t恤，，，嗯~

2022-07-21 15:35 1 回复

我们要去北方 算法胎教

2022-07-22 07:42 回复

吃饭睡觉打逗逗2

从业八年的社畜一枚，很羡慕现在的年轻人，有各种网课，各种免费质量好的课程，各种免费的解决方案。

和其他课程不一样的是，104 和其他games课程不一样，因为并不是“老师背景的人”，明显感觉一个很贴地气的“社畜”，毫无保留的把知道的东西奉献出来。

目前工作比较忙，看视频直播有时候会忘记，一般就是早八点起床，看一个小时视频，中午吃饭看剩下的一半，晚上回来在看剩下的，周末抽时间研究引擎。2333，想要体恤，拿出来装逼

大马克一只

建议整一个游戏引擎业的黑话集锦然后聚在一起印文化衫背后，比如炸模穿模彼得潘

2022-07-21 15:20 回复

Rivers小何

刚开始看104的时候我还是学生，，，现在已经是社畜了既然有机会拿到周边，就正好一并感谢课题组以及身边的人吧

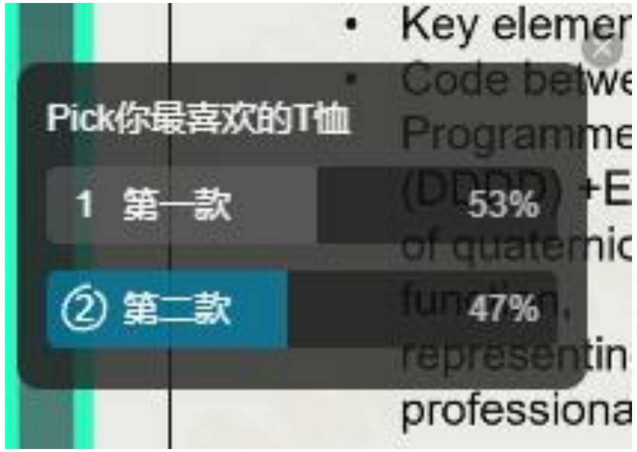
Voices from Community – T-shirt Style Voting

- Voting results for T-shirt style: 53% vs. 47% (119 total)

- We will make Style 1 T-shirts as souvenirs

- Please comment under “Lecture 17” @Bilibili before 24:00 this Sunday 7/31 about your great ideas of what souvenirs you want for Piccolo

- We will give out 10 T-shirts for the best comments



Rewarding list for last event:

@吃饭睡觉打逗逗2, @地地地瓜瓜大王, @Rivers小何,

@微夏丿风, @亻Biu, @红魔族第一的程序猿,

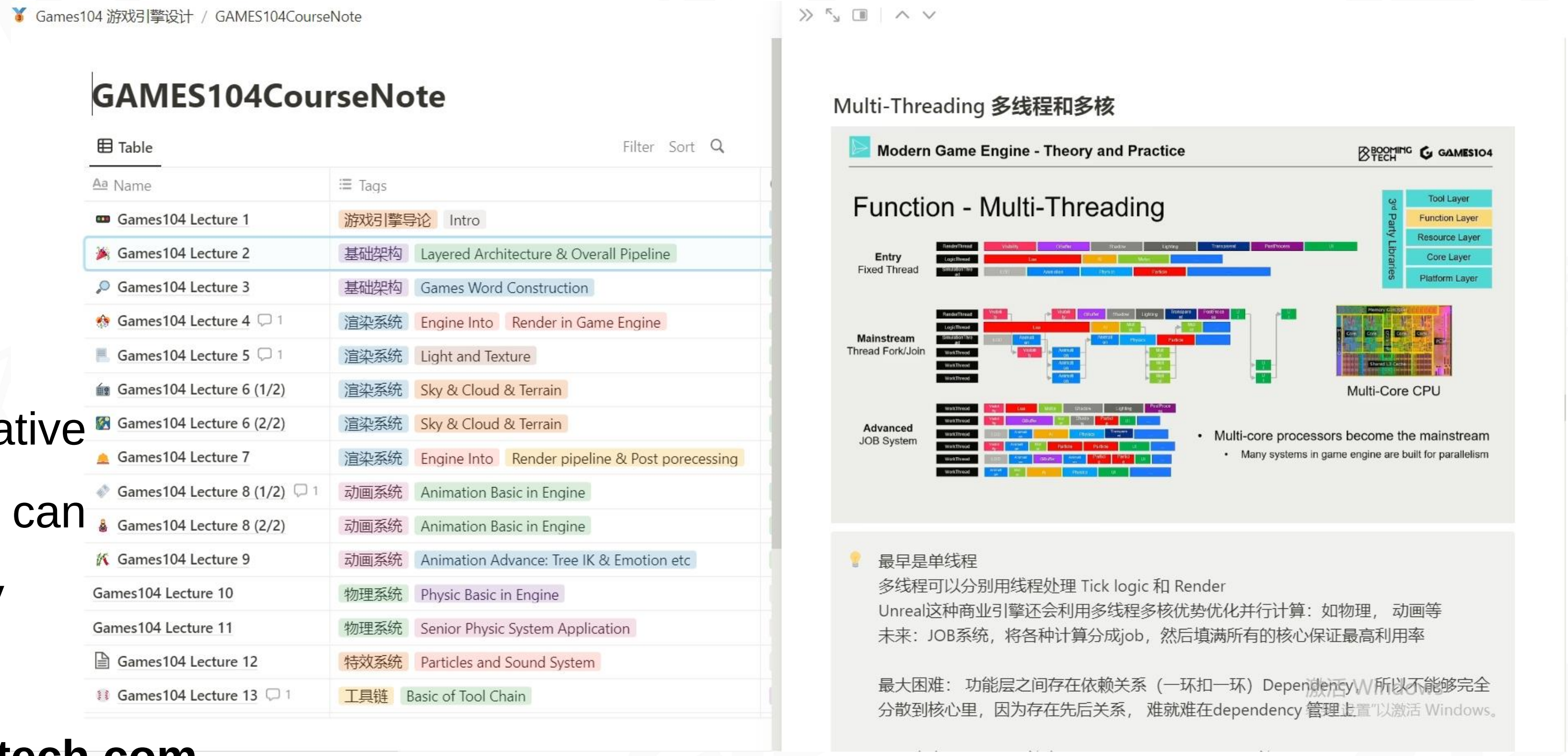
@川明177, @Quincy-1, @宠花上天, @AsEiif

Voices from Community – Great Lecture Notes

- Really appreciated for sharing notes and making contributions to Piccolo community

- If you have other lecture notes, creative ideas, or any interesting projects, we can help you to release in our community

- contact email:
- **piccolo-gameengine@boomingtech.com**



@主题歌的一天

Q&A

- Q1: What’s your opinion on AI reading operating instructions from players directly?
- Q2: What’s the budget of computation of AI behaviors?
- Q3: Is it feasible to distribute the computation of AI agents over the network to alleviate performance pressure from AI systems?

Lecture 16

Gameplay Systems

Advanced Artificial Intelligence

WANG XI

GAMES 104

2022

Outline of Artificial Intelligence Systems

01.

AI Basic

- Navigation
- Steering
- Crowd Simulation
- Sensing
- Classic Decision Making Algorithms

02.

Advanced AI

- Hierarchical Tasks Network
- Goal-Oriented Action Planning
- Monte Carlo Tree Search
- Machine Learning Basic
- Build Advanced Game AI

Hierarchical Tasks Network

Overview

HTN assumes there are many **Hierarchical tasks**

Transformers fall of cybertron(2012)

Make a Plan Like Human

Hierarchical:

- people in real world usually make their plan hierarchically

Take a class

Decompose

- Prepare the material
- Go to the classroom
- Learn
- Q&A after class

Select

Method 1

- Find Books in Library
- Borrow Books

Method 2

- Search on Internet
- Download and Print

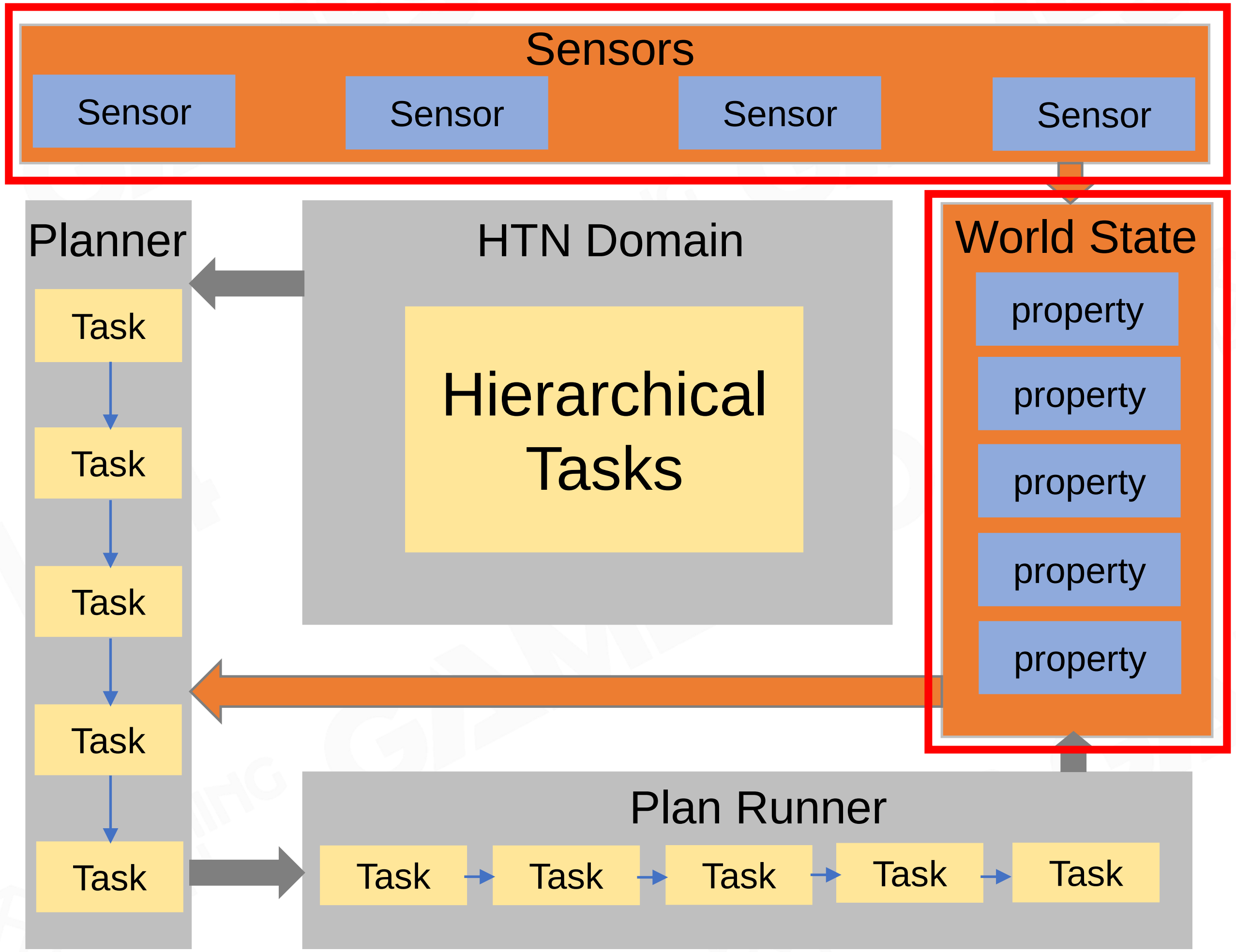
HTN Framework (1/2)

World state

- Contains **a bunch of properties**
- **Input to planner**, reflect the status of world
- It's a **Subject World View in AI Brain**

Sensors

- Perceive changes of environment and **modify world state**
- **It's more like Perception**

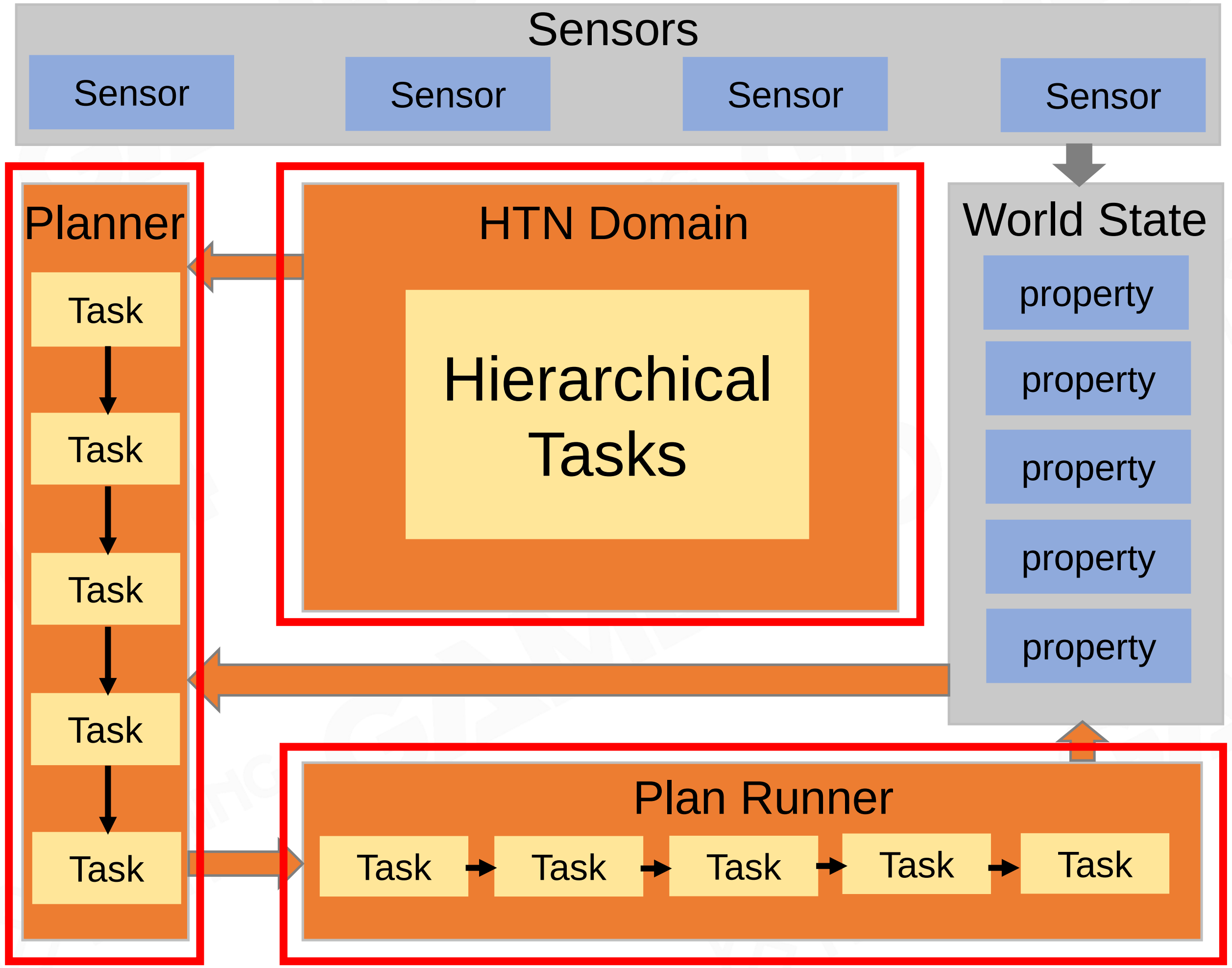


HTN Framework (2/2)

HTN Domain
- Load from asset
- Describe the relationship of **hierarchical tasks**

Planner
- **Make a plan** from World State and HTN Domain

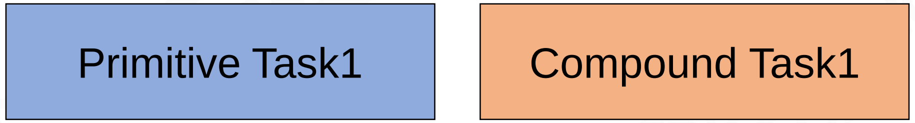
Plan Runner
- Running the plan
- **Update the world state** after the task



HTN Task Types

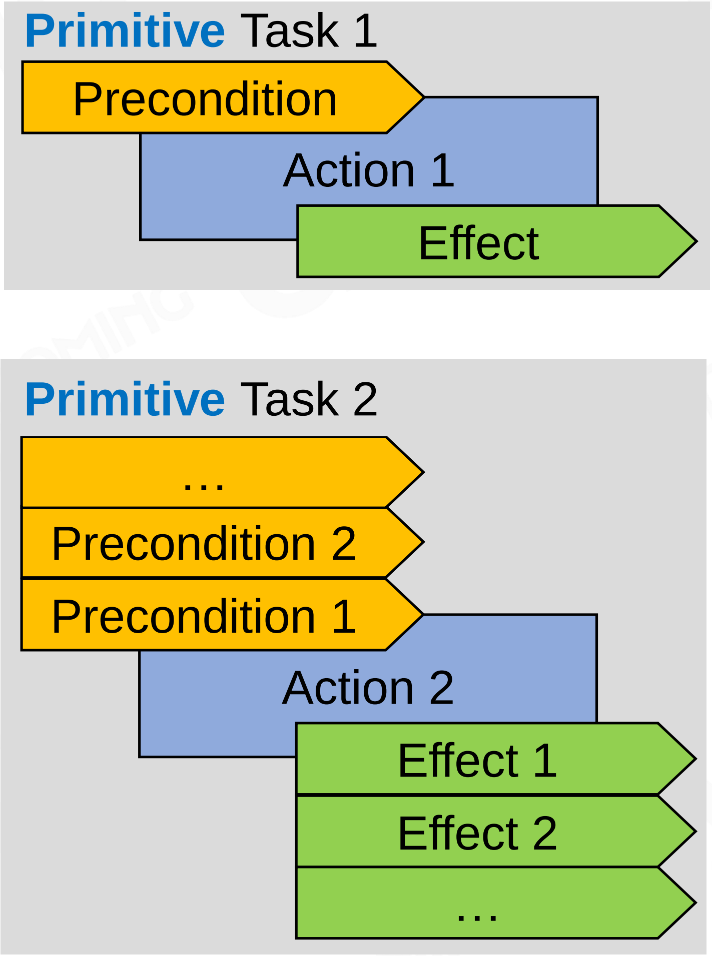
Two types of Tasks

- Primitive Task
- Compound Task

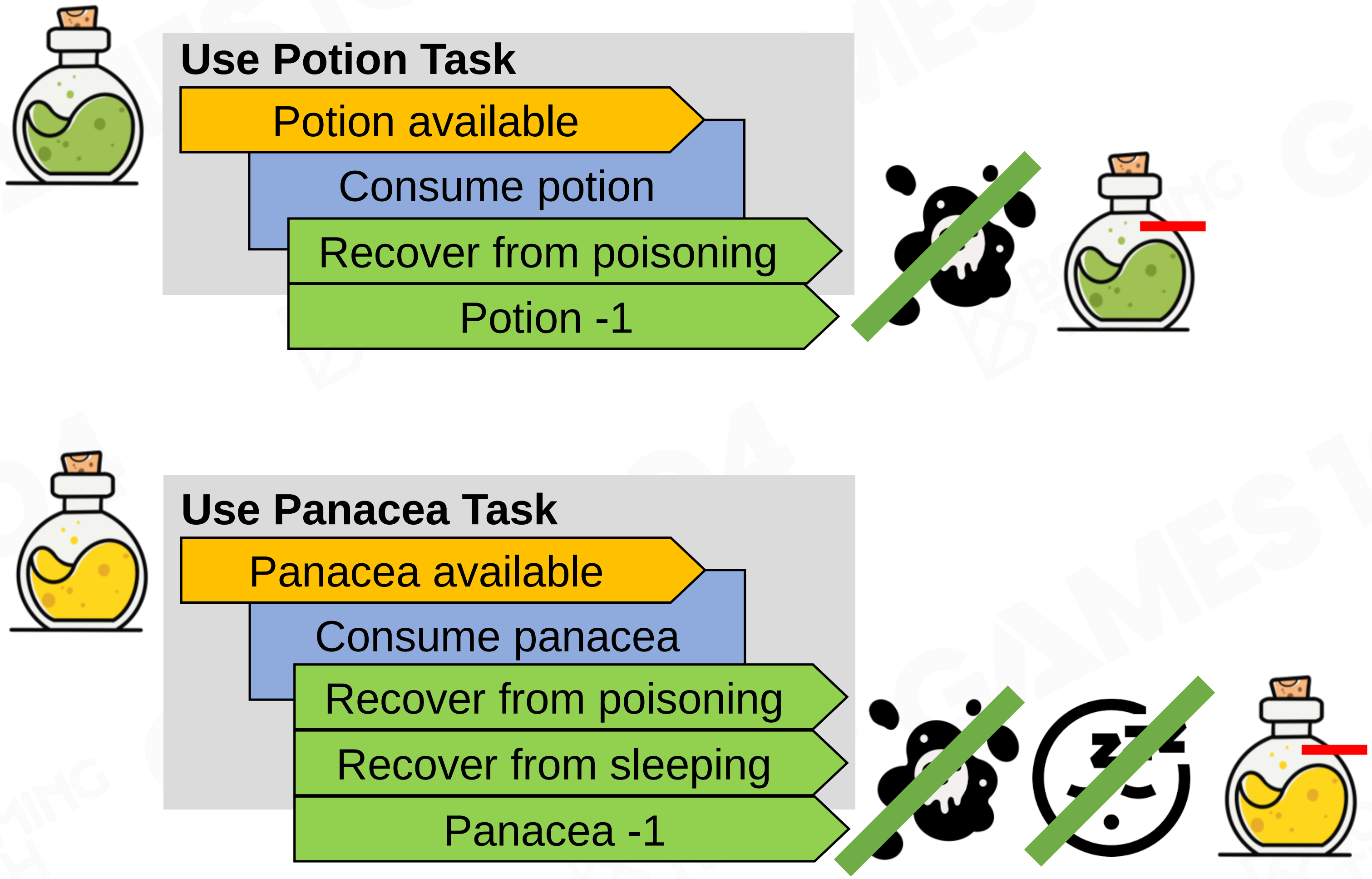


Primitive Task (1/2)

- Preconditions
  - Determine whether an action could be executed
  - Check whether **properties of game world** being satisfied
- Action
  - Determine what action the primitive task executes
- Effects
  - Describe how the primitive task **modify the game world state properties**



Primitive Task (2/2)



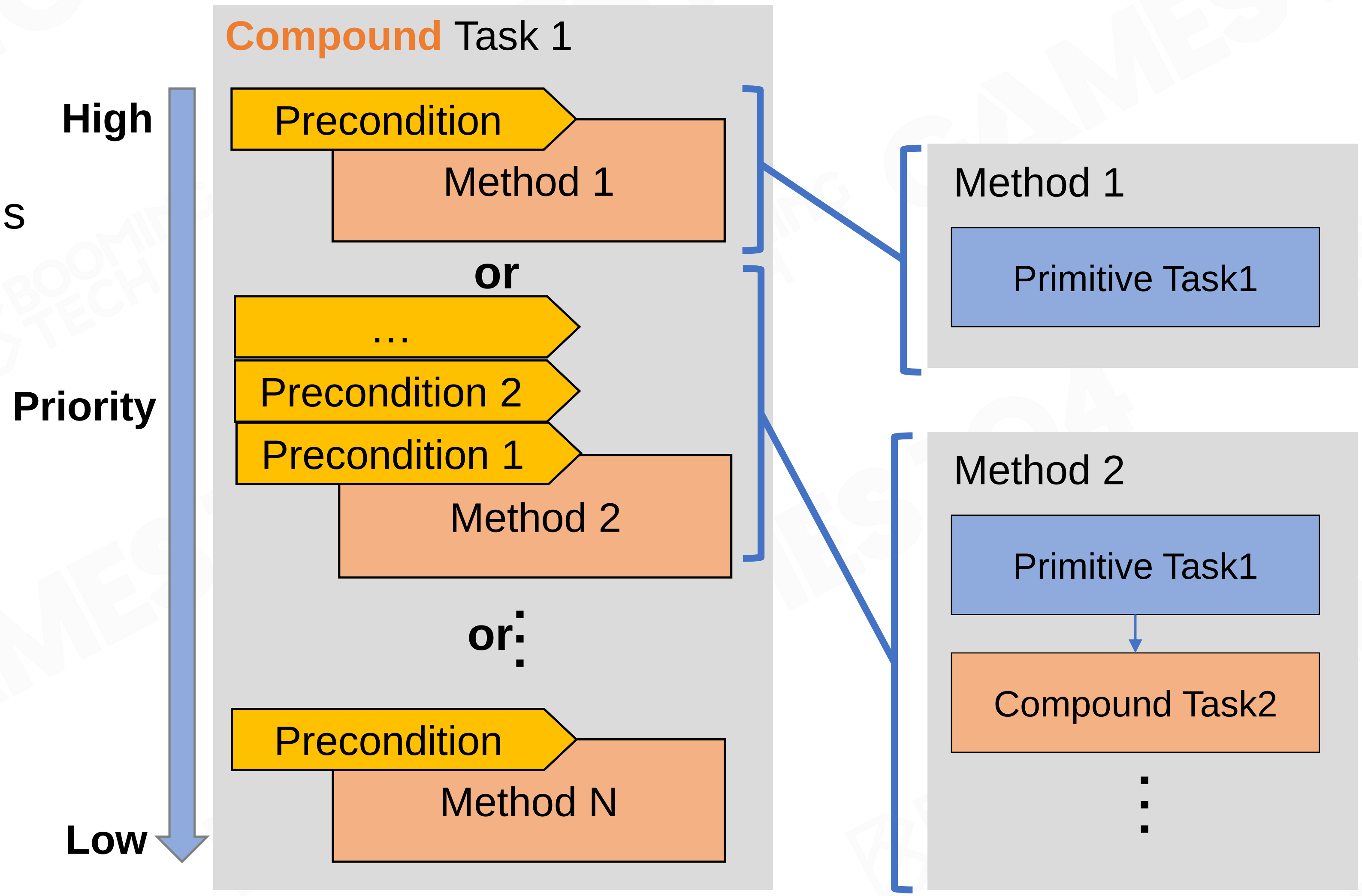
Compound Task (1/2)

Compound Tasks

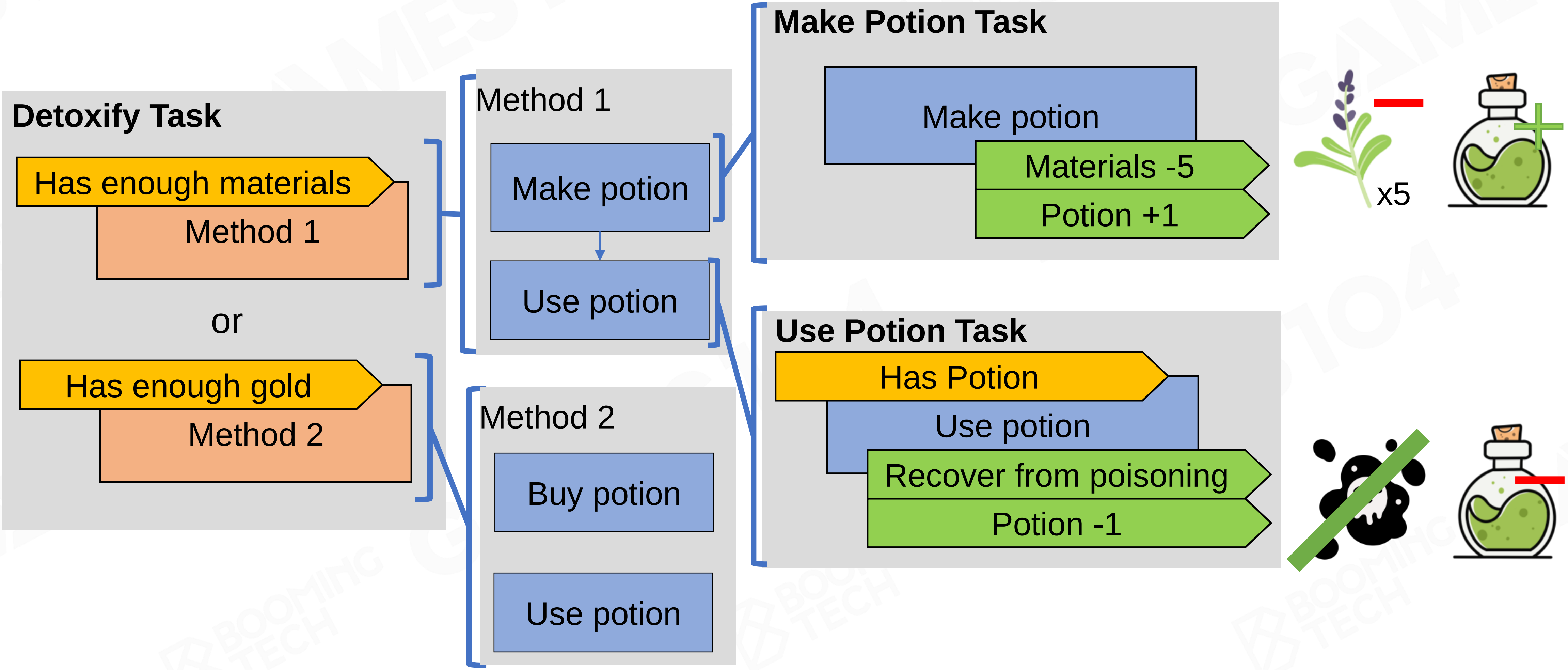
- Contain several methods
- Methods have different priority
- Each method has preconditions

Method

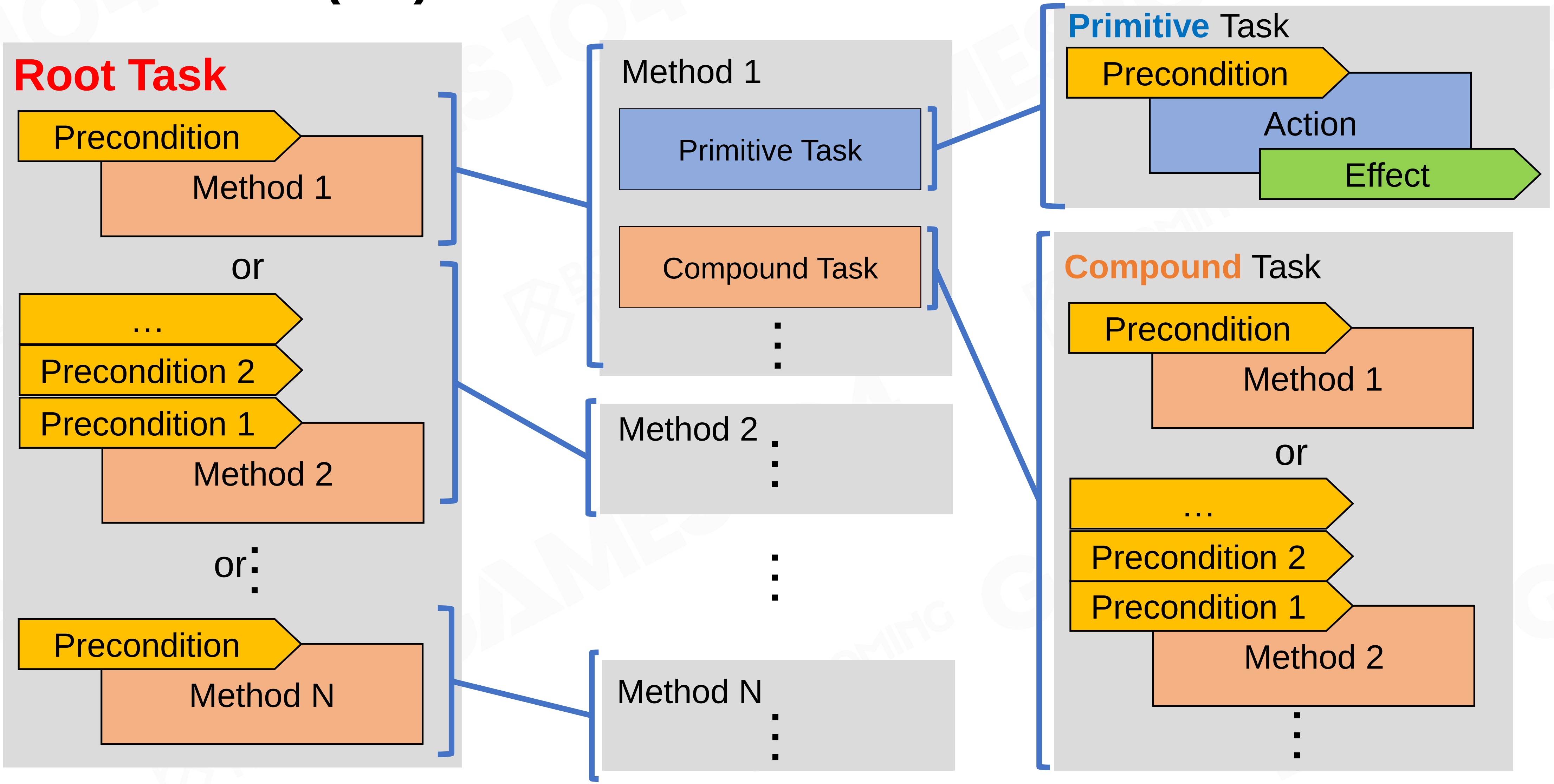
- contains a chain of sub-Tasks
- Sub-task could be a **primitive task** or a **compound task**



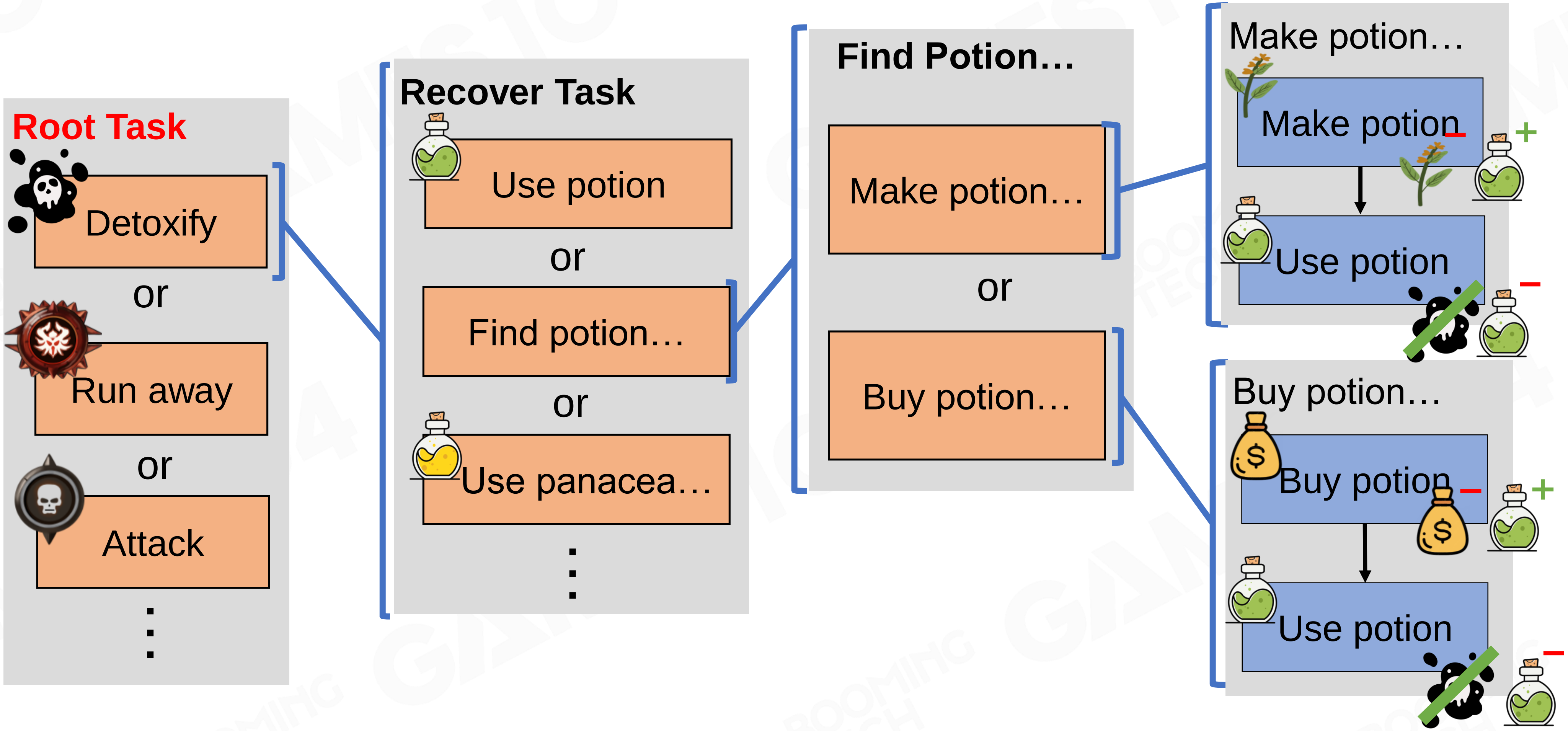
Compound Task (2/2)



HTN Domain (1/2)



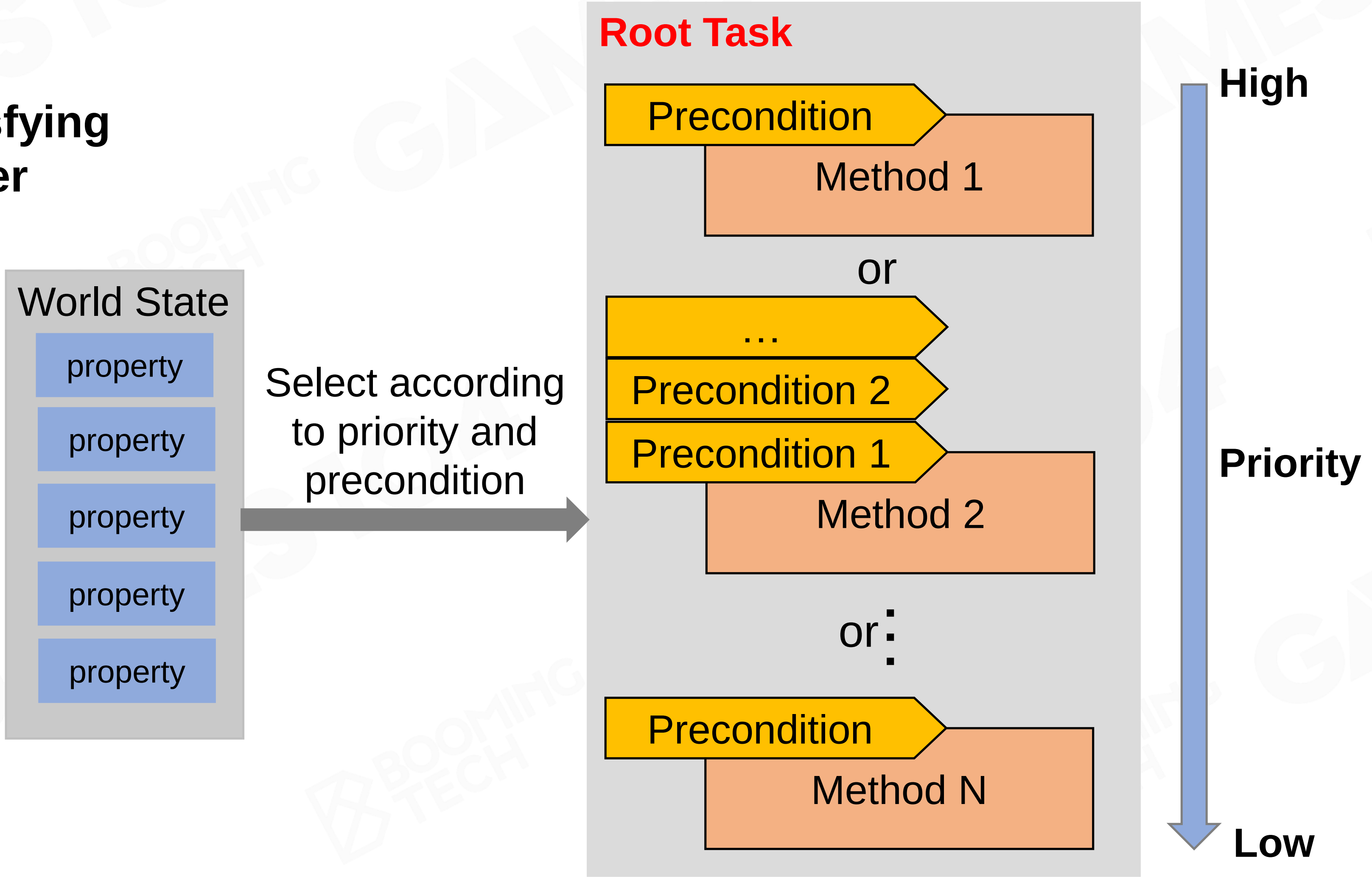
HTN Domain (2/2)



Planning (1/7)

Step 1

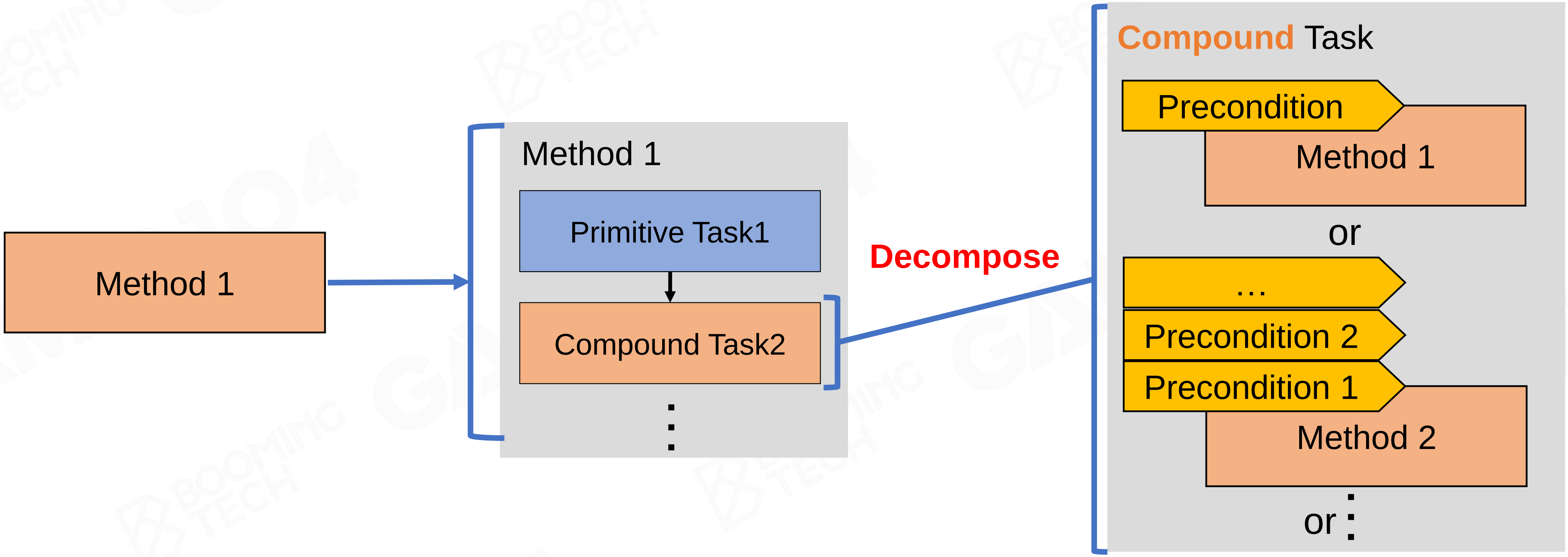
- Start from the **root task**
- Choose the method **satisfying the precondition in order**



Planning (2/7)

Step 2

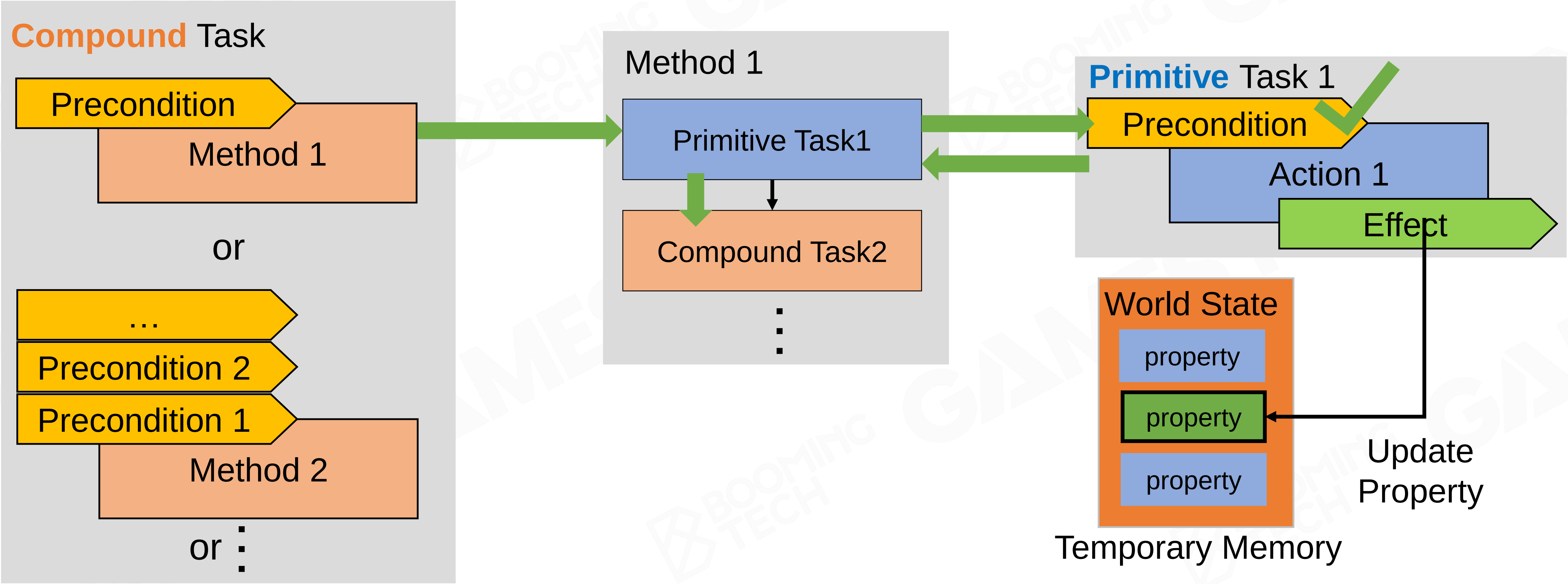
- Decompose the method to tasks
- Check precondition in order
- Decompose the task if it is a compound task



Planning (3/7)

Step 2 (For primitive tasks)

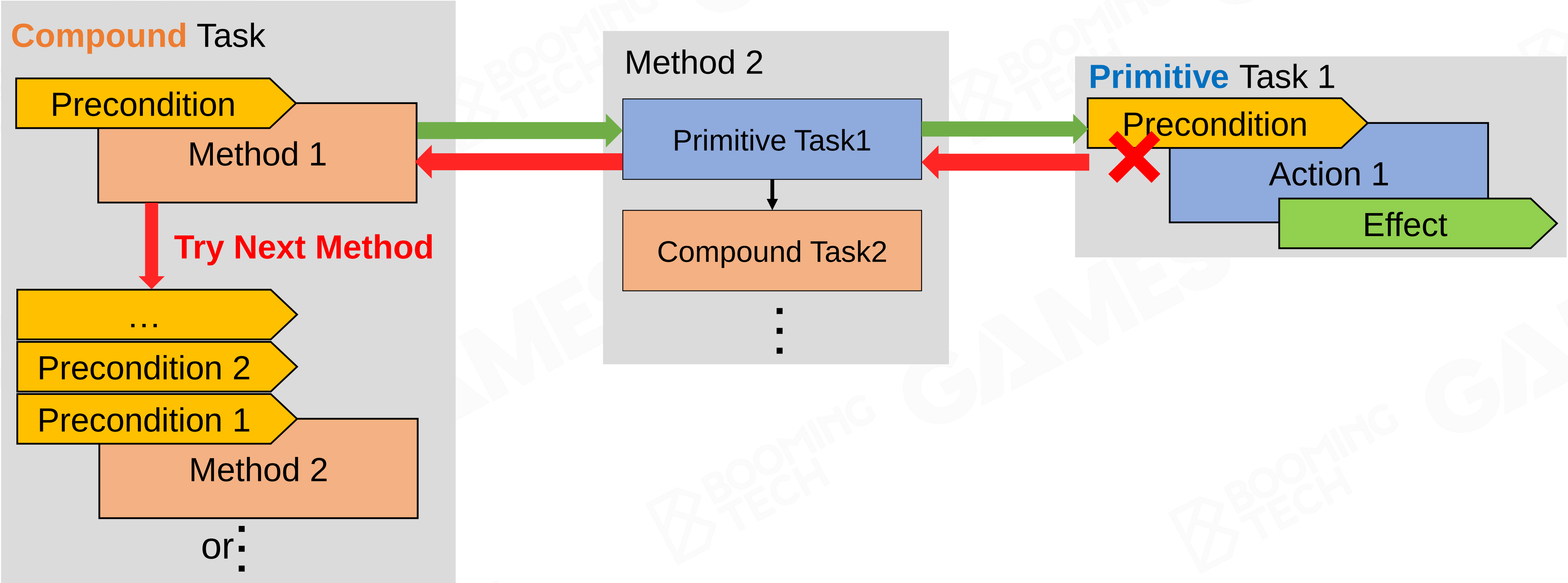
- Assume all action will be succeed, update "world state" in temporary memory
- World state has a duplicated copy in planning phase for scratch paper



Planning (4/7)

Step 2 (For primitive tasks)

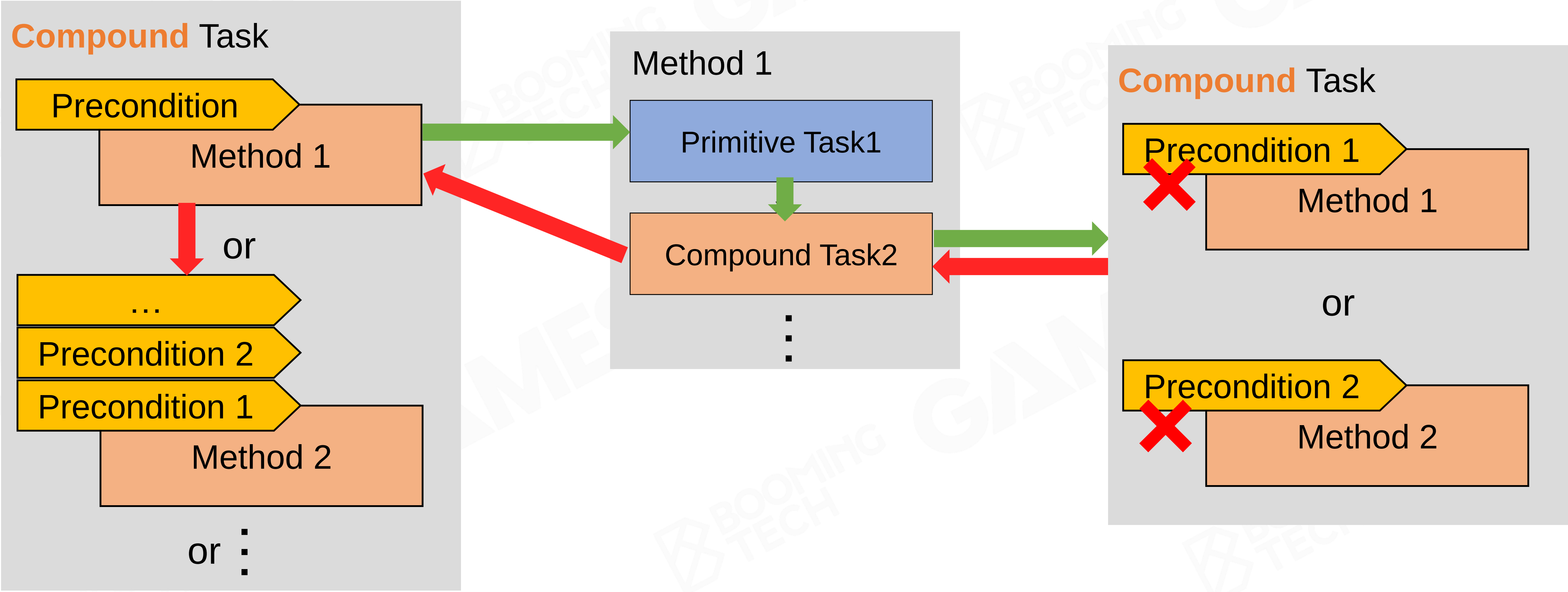
- go back and select a new method if precondition is not satisfied



Planning (5/7)

Step 2 (For compound task)

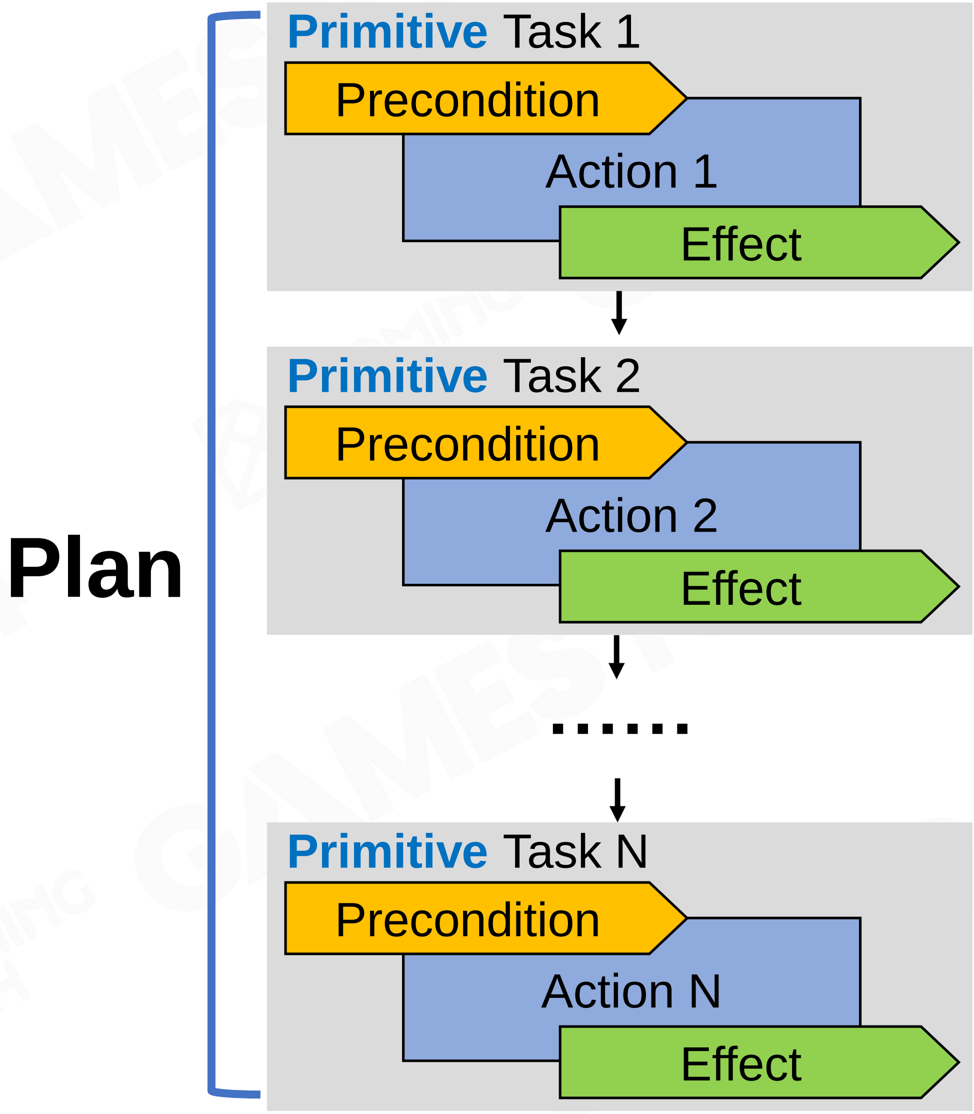
- select the next method if precondition is not satisfied



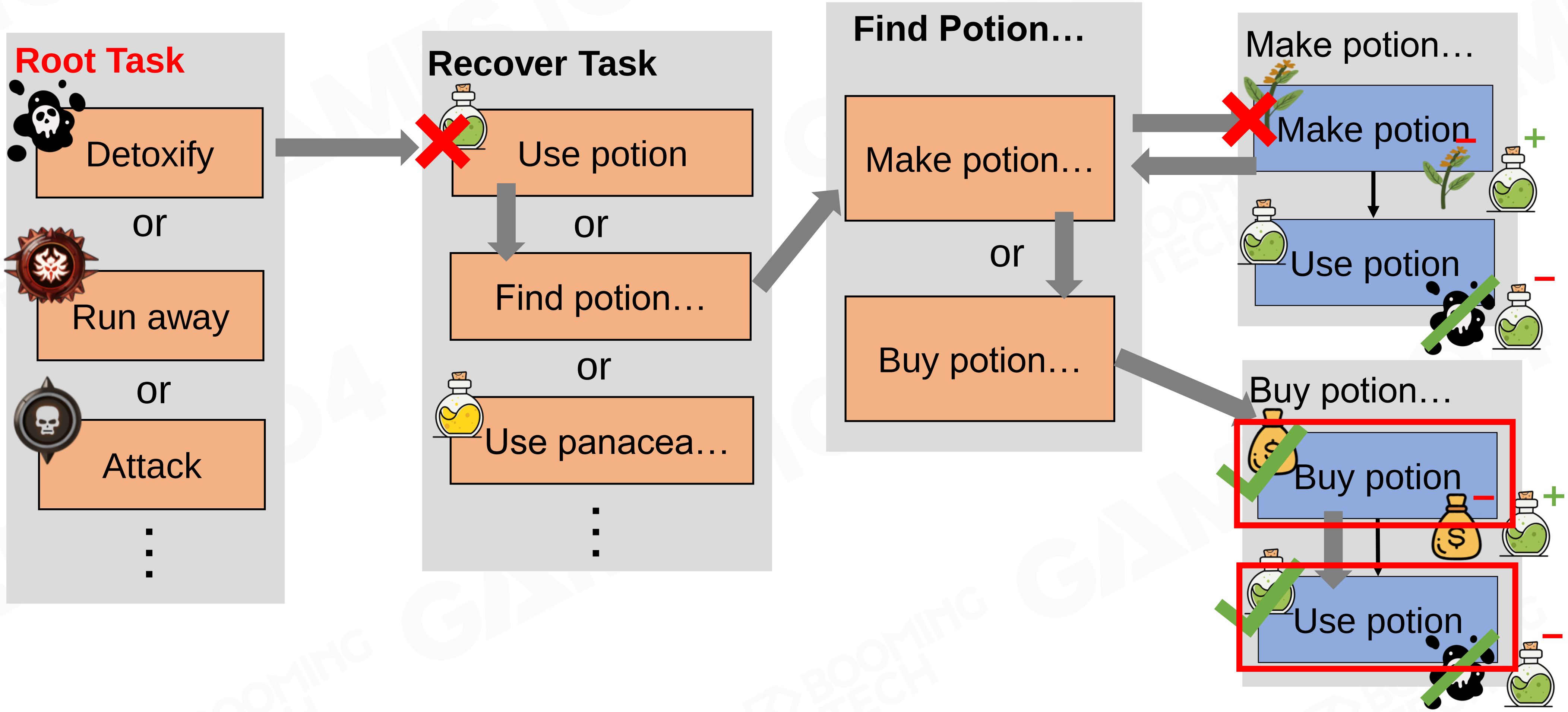
Planning (6/7)

Step 3

- Repeat step 2 until no more task needs to be done
- The final plan contains only primitive tasks



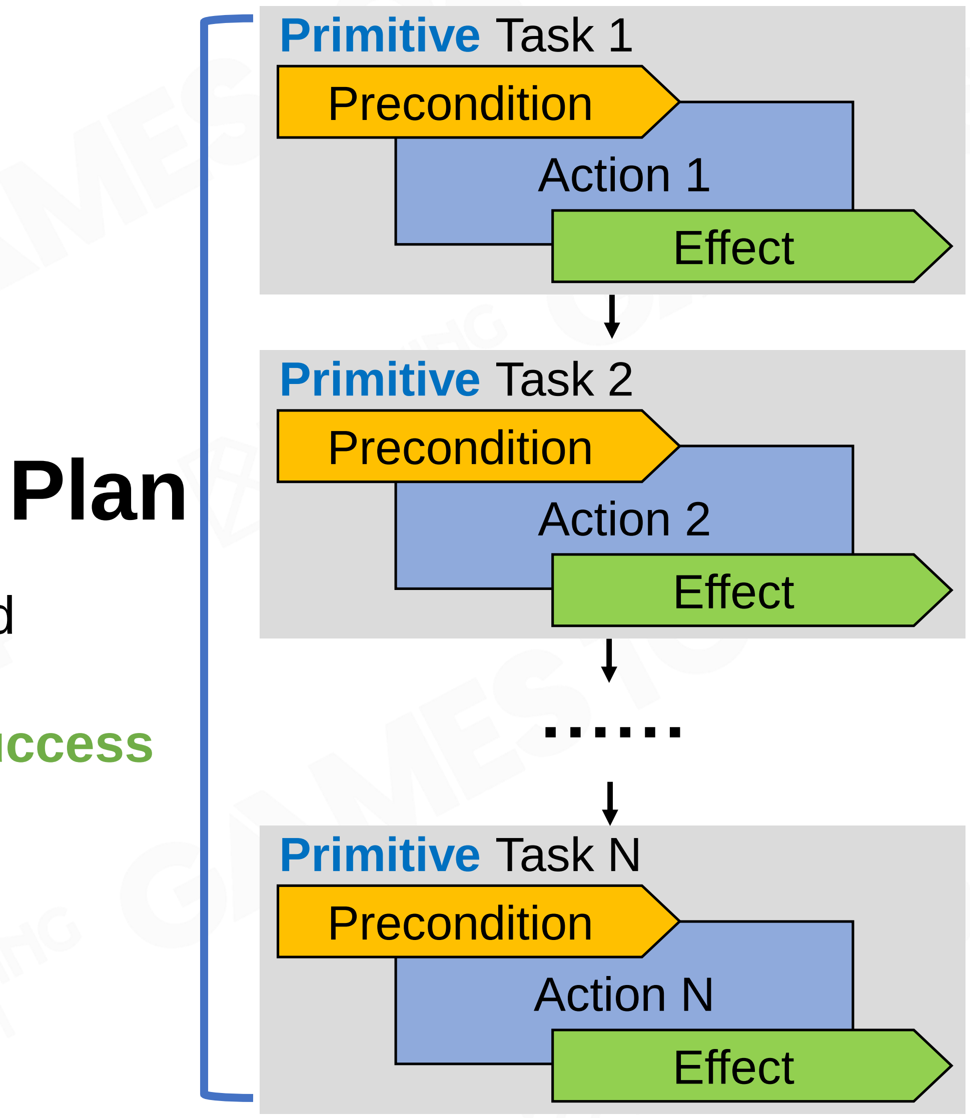
Planning (7/7)


Root Task
Detoxify
or
Run away
or
Attack
⋮
Recover Task
Use potion
or
Find potion…
or
Use panacea…
⋮
Find Potion…
Make potion…
or
Buy potion…
Make potion…
Make potion
Use potion
Buy potion…
Buy potion
Use potion

Run plan

Run plan
- Execute tasks in order
- Stop until **all tasks succeed**, or **one task failed**

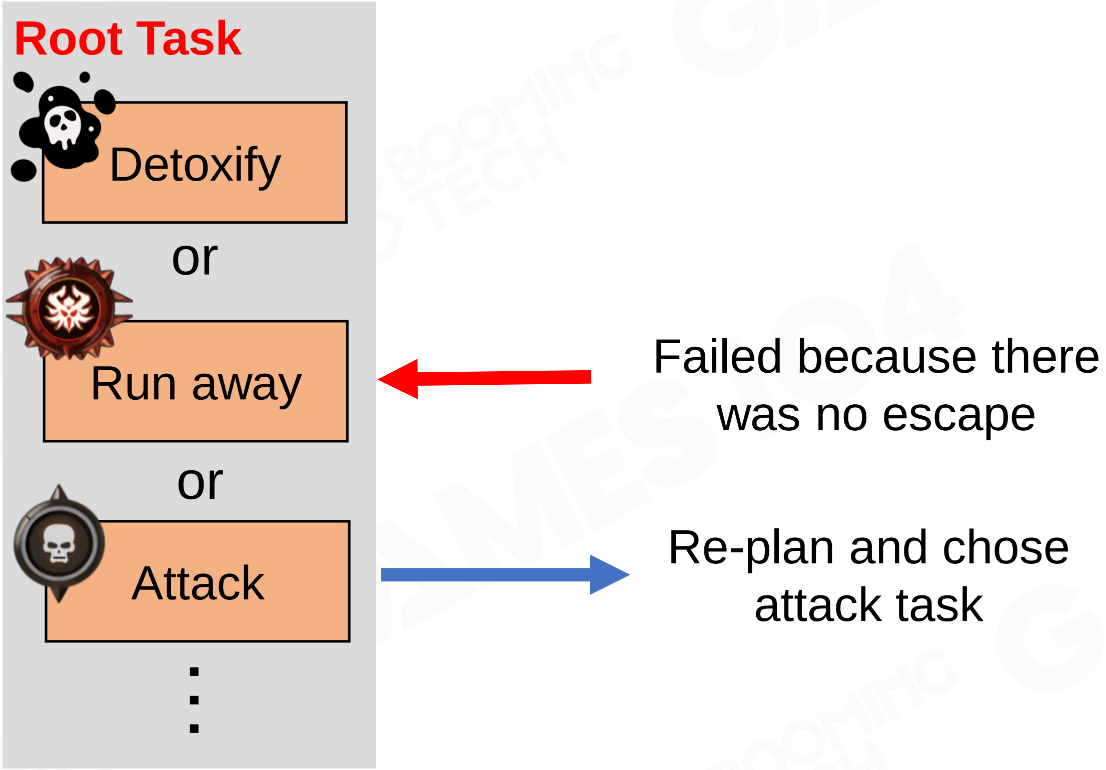
Execute task
- Check precondition and return **failure** if not satisfied
- Execute action
  - if succeed -> **update world state** and return **success**
  - if failed -> return **failure**



Replan

There are three situations that the agent could start plan

- Not have a plan
- The current plan is finished or failed
- The World State changes **via its sensor**



Conclusion

Pros:
- HTN is similar with BT, and it is **more high-level**
- It outputs a plan which has **long-term** effect
- It would be **faster** compared to the BT in the same case

Cons:
- Player’s behavior is unpredictable, so the tasks **may be easy to fail**
- The World state and the effect of tasks are **challenging for designers**

Goal-Oriented Action Planning

Goal-Oriented Action Planning (GOAP)

- **GOAP** is more **automated**
- It takes **backward planning** rather than forward

Assassins Creed Odyssey

Structure

Sensors and **World State**

- Similar to HTN

Goal set

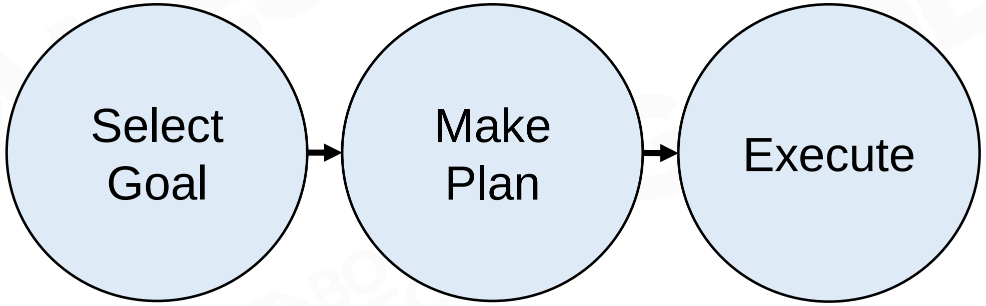
- All available goals

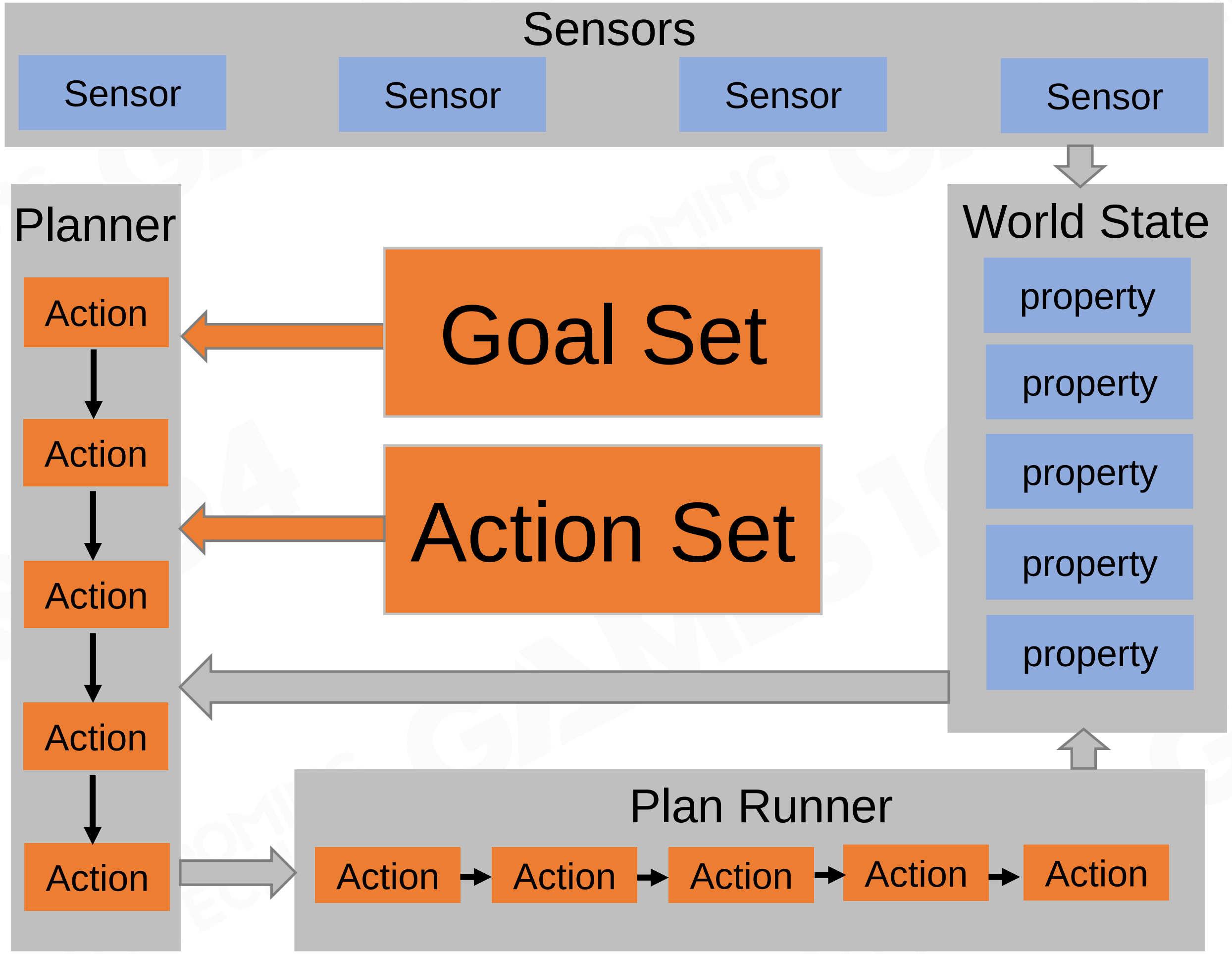
Action set

- All available actions

Planning

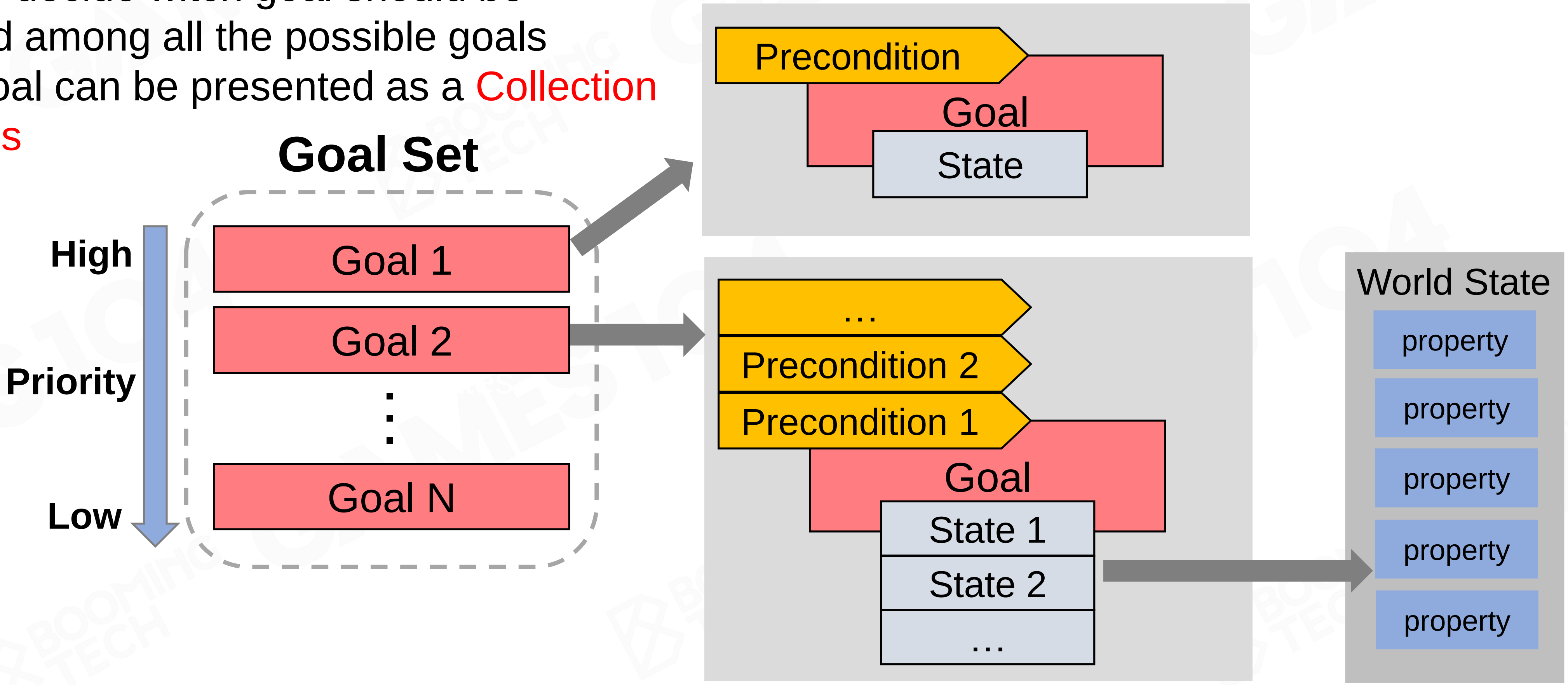
- Output sequence of actions



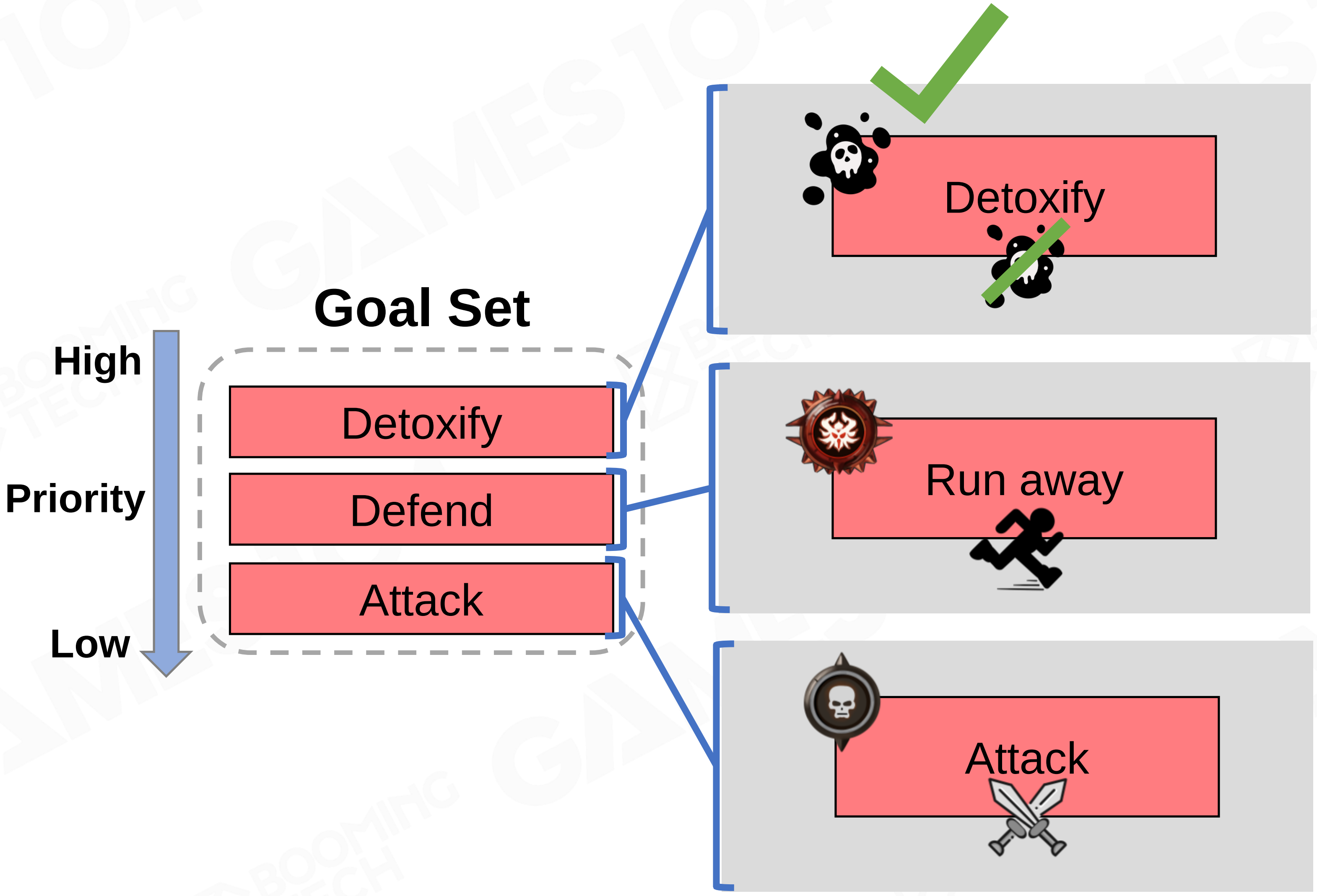


Goal Set

- **Precondition** decides witch goal will be selected
- **Priority** decide witch goal should be selected among all the possible goals
- Each goal can be presented as a Collection of States



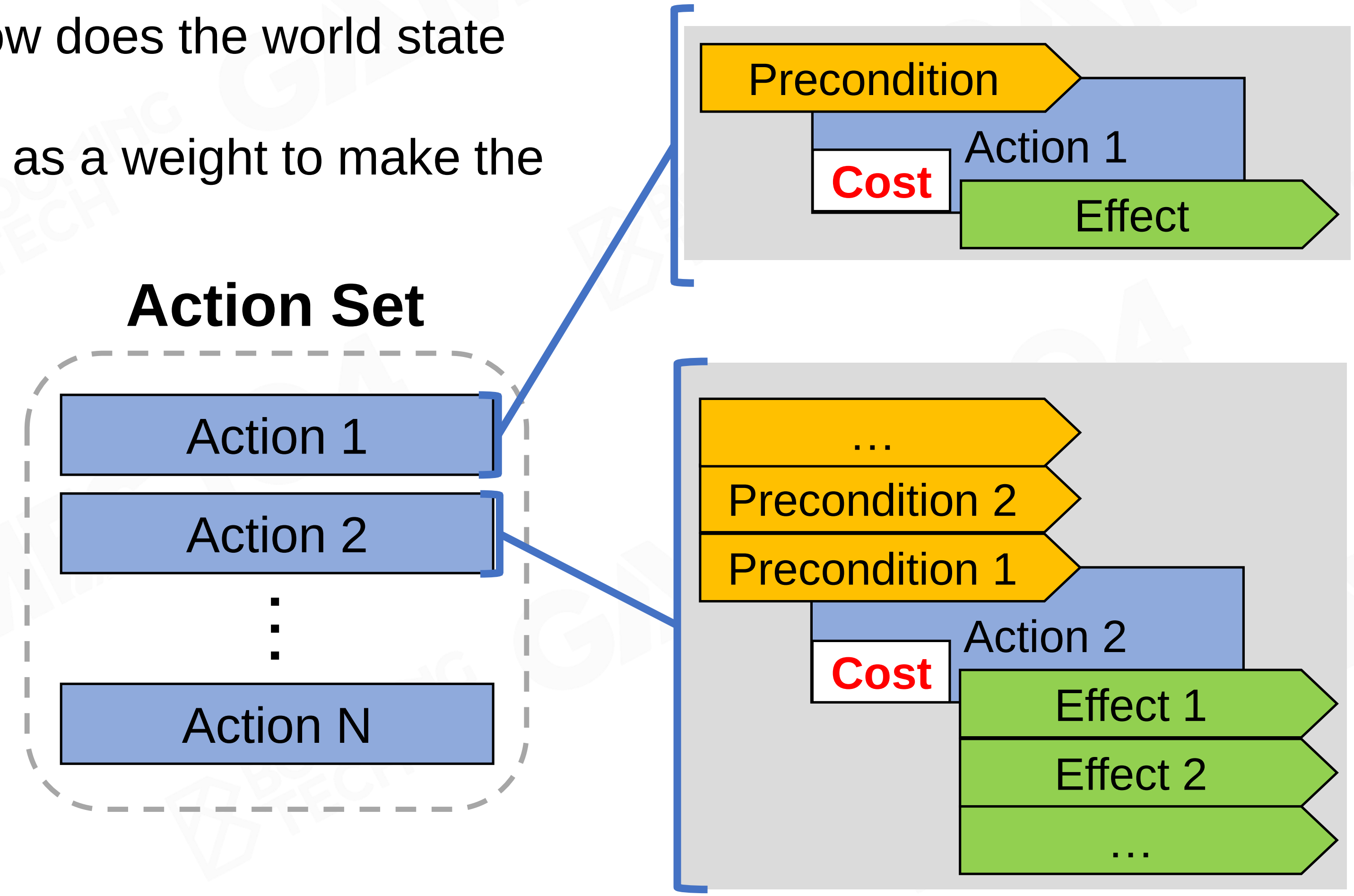
Goal Selection



Action Set

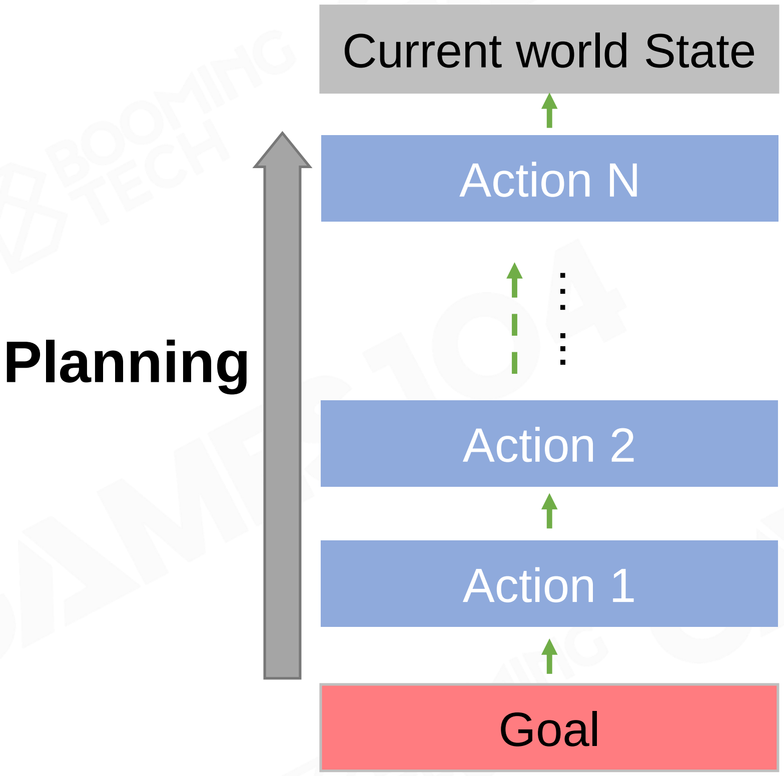
Action in GOAP is with precondition, effect and cost

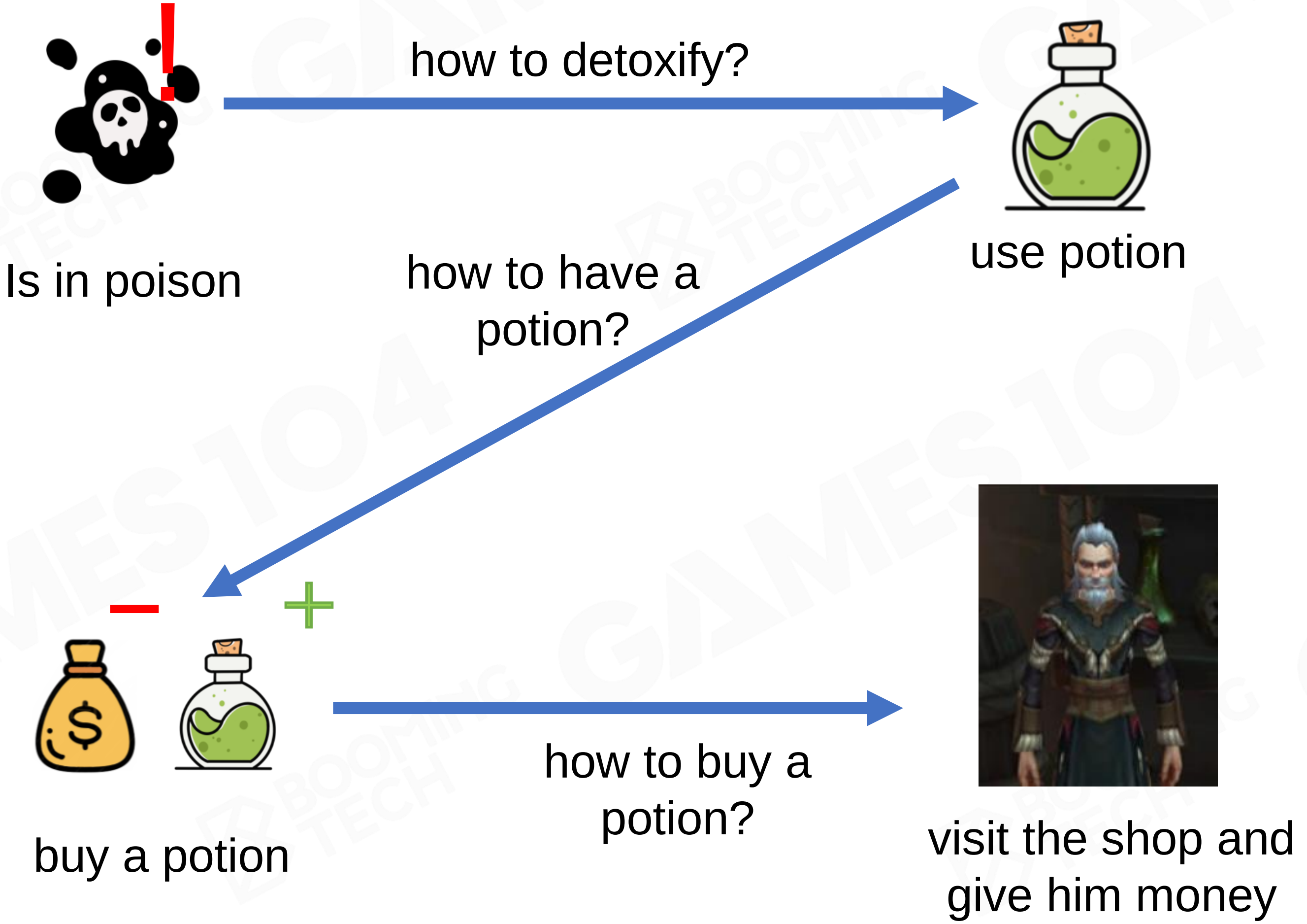
- Precondition: in which state, character can do this action
- Effect: after the action is done, how does the world state changes
- **Cost**: defined by developer, used as a weight to make the plan which has the lowest cost



Backward Planning Like a Human

- When making a plan, start from goal state

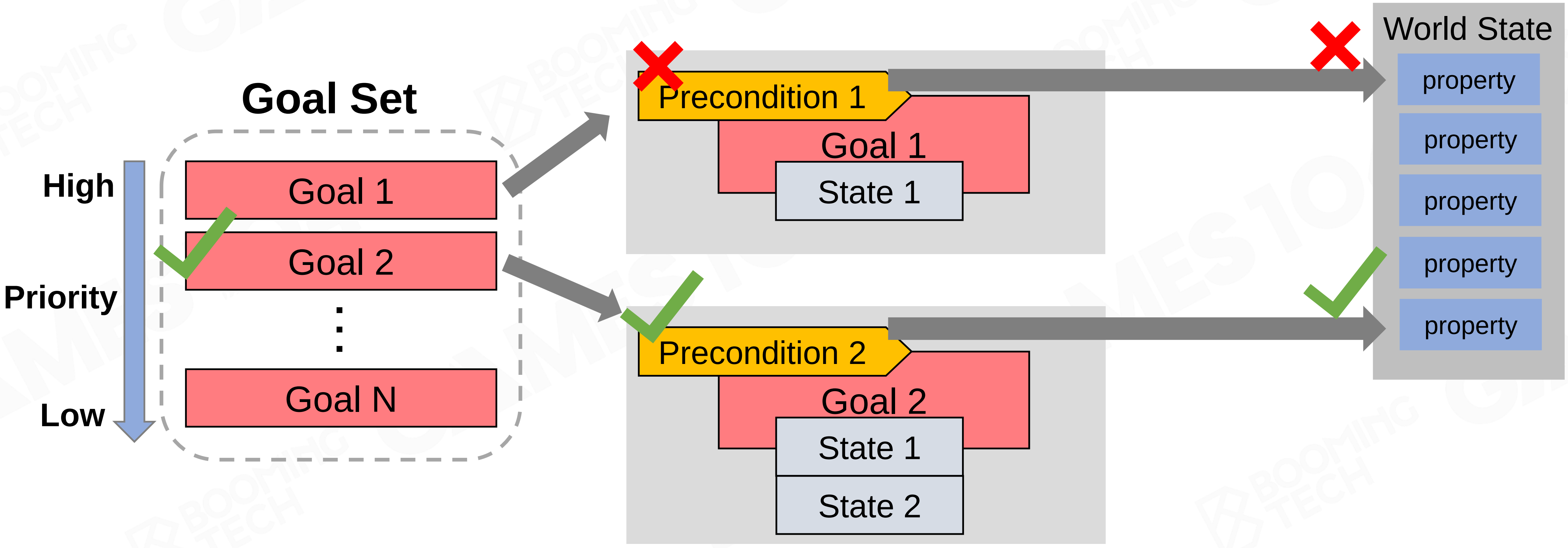




Planning (1/4)

Step 1

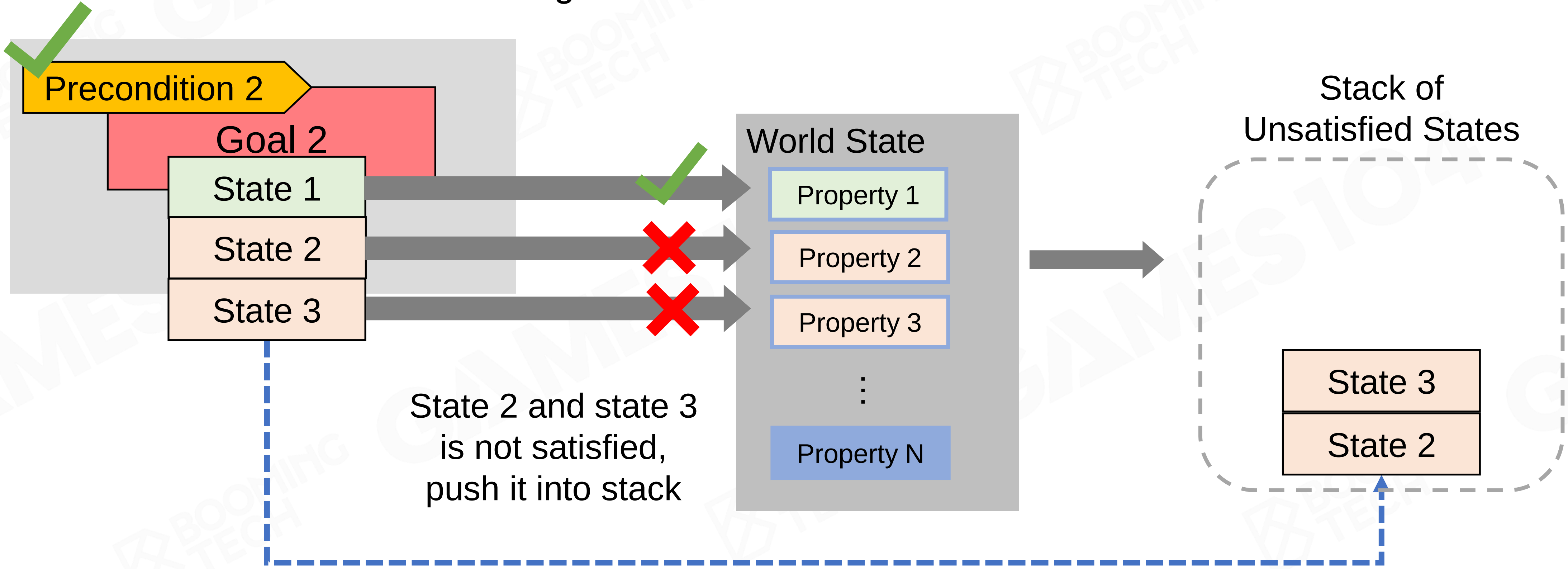
- Check goals according to priority
- Find the first goal of which precondition is satisfied



Planning (2/4)

Step 2

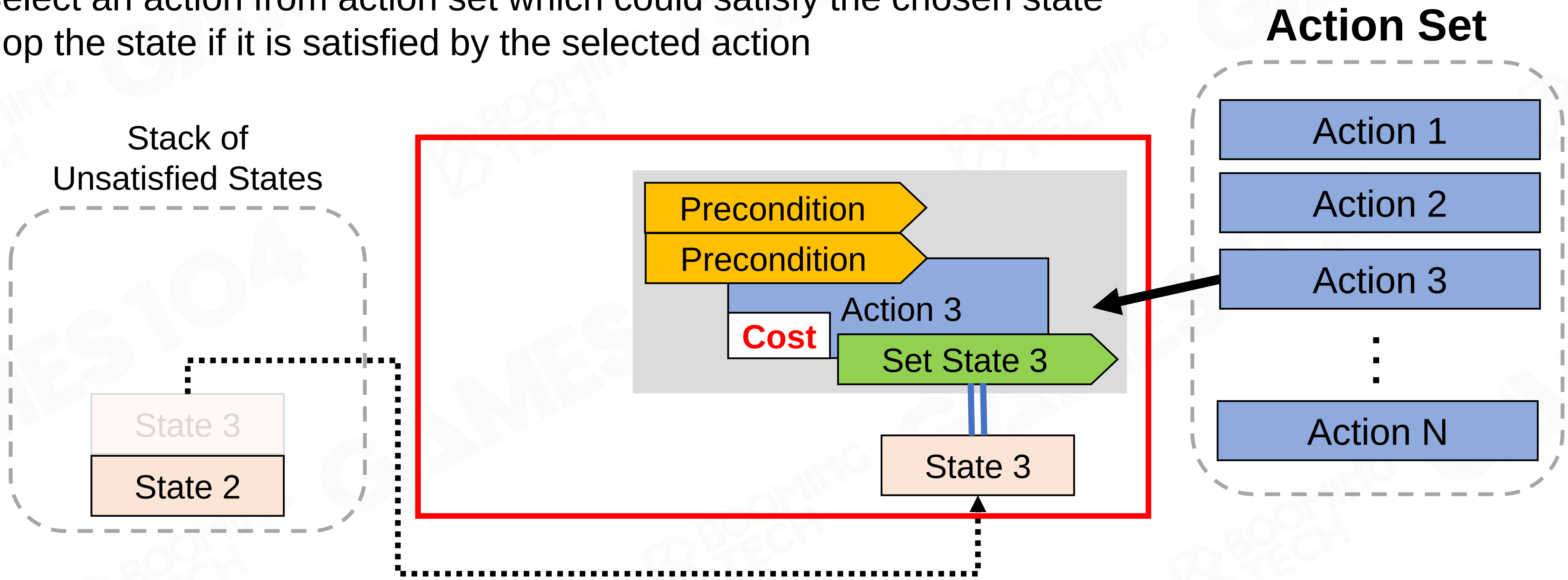
- Compare the target state with world state to find unsatisfied goal
- Set all unsatisfied states of the goal into a stack



Planning (3/4)

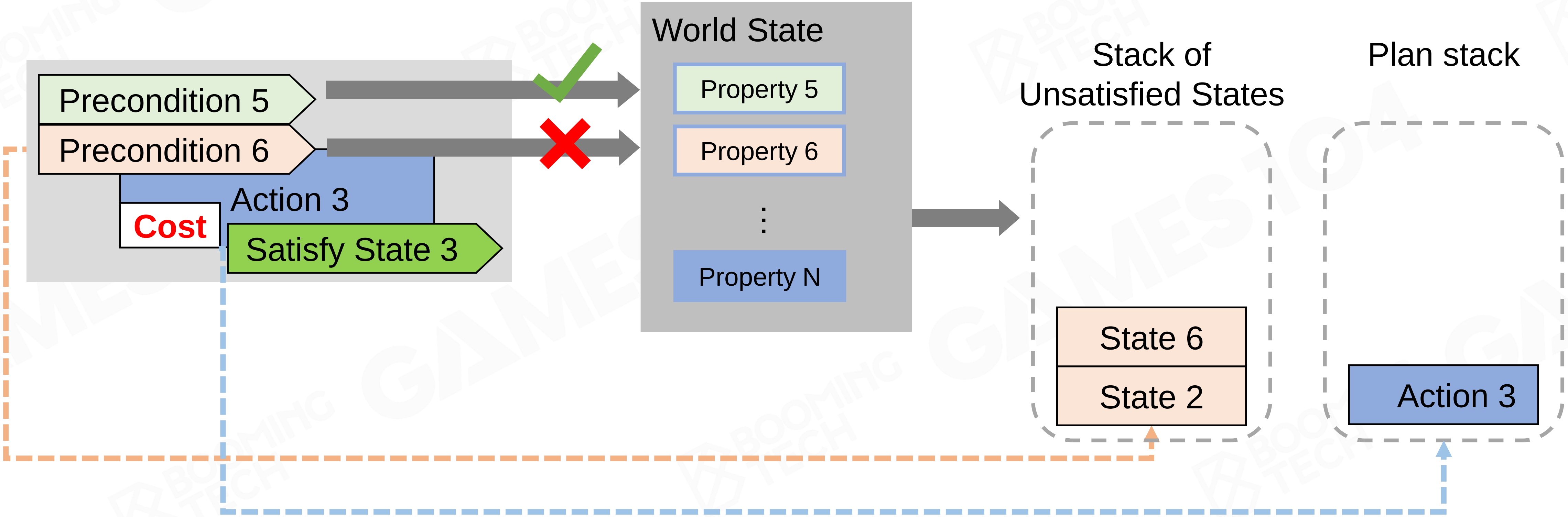
Step 3

- Check the top unsatisfied state from the stack
- Select an action from action set which could satisfy the chosen state
- Pop the state if it is satisfied by the selected action



Planning (4/4)

Step 4
- Push action to plan stack
- Check precondition of corresponded action
- If precondition is not satisfied, push state to stack of unsatisfied states



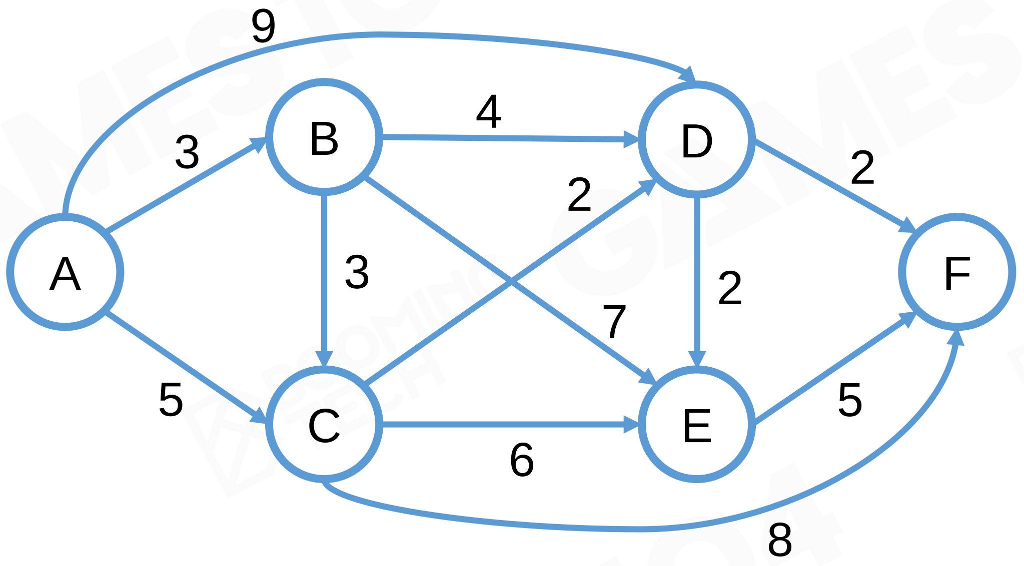
Build States-Action-Cost Graph

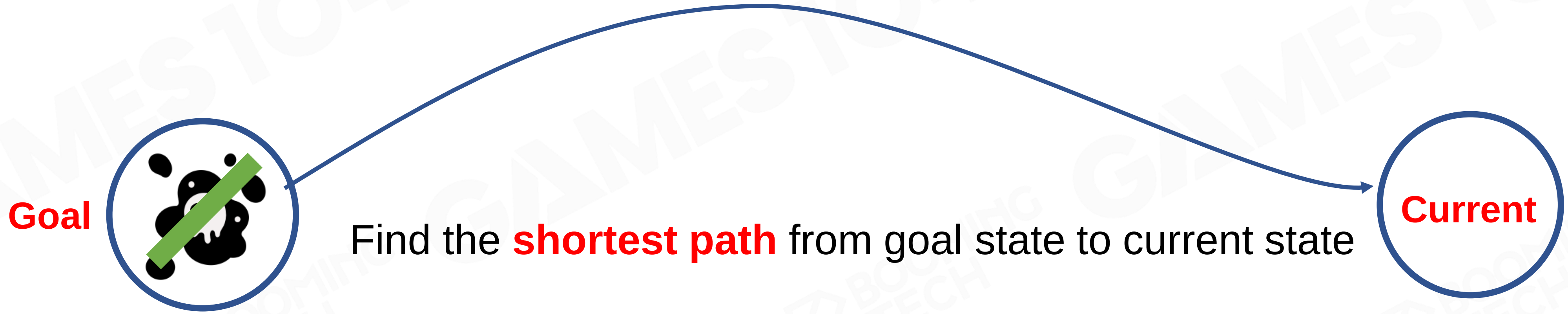
Can be turned into a path planning problem

- Node : **Combination of states**
- Edge : Action
- Distance : Cost

Search direction

- Start node : states of the goal
- End node : current states

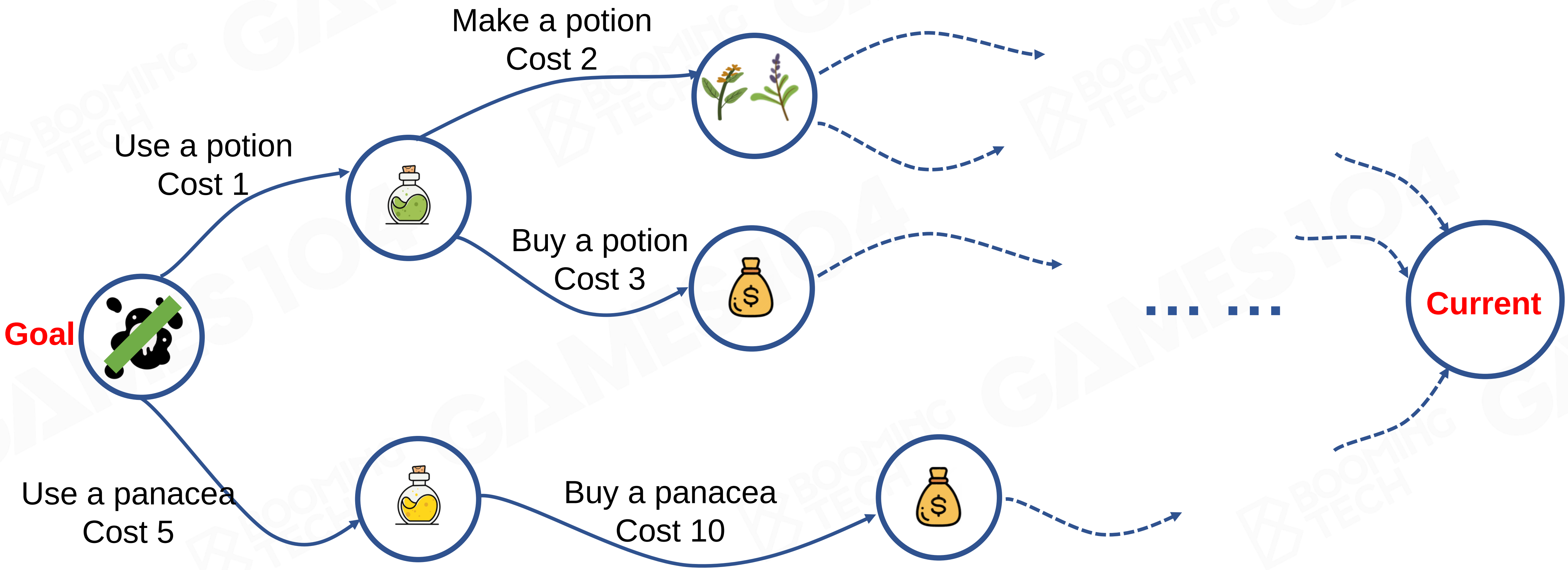




The Lowest Cost Path

Can use A* or other shortest path algorithms

- Heuristics can be represented with number of unsatisfied states



Conclusion

Pros:

- Compared with HTN, GOAP plans is more dynamic
- Decoupling goals and behaviors
- HTN can easily make precondition/effect mismatching mistakes

Cons:

- In a single AI system, the runtime planning would be slower than BT/FSM/HTN
- Also needs a well-represented world state and action effect

Monte Carlo Tree Search

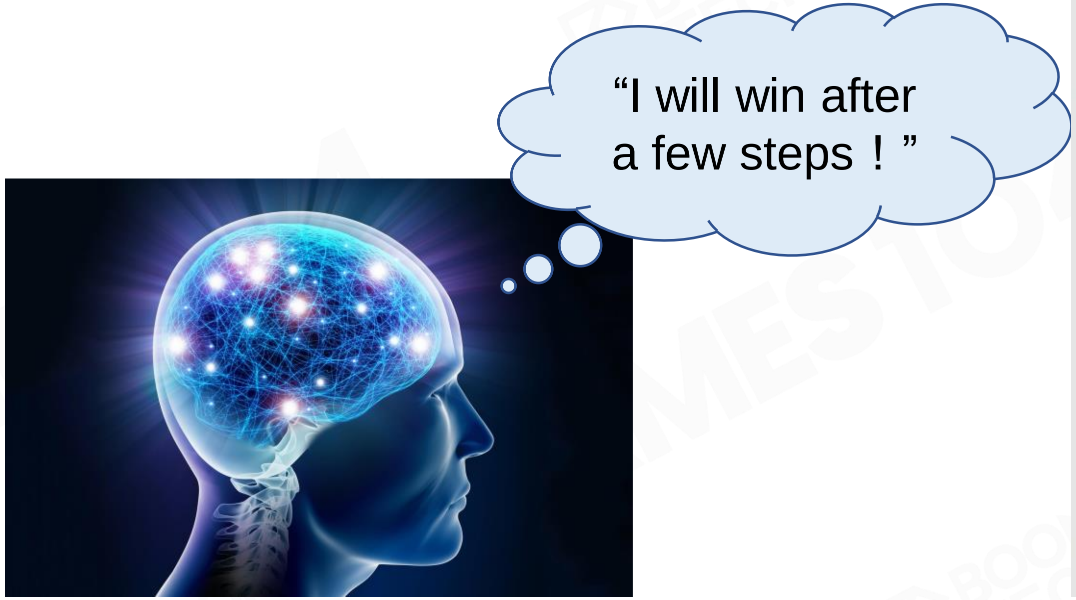
Monte Carlo Tree Search

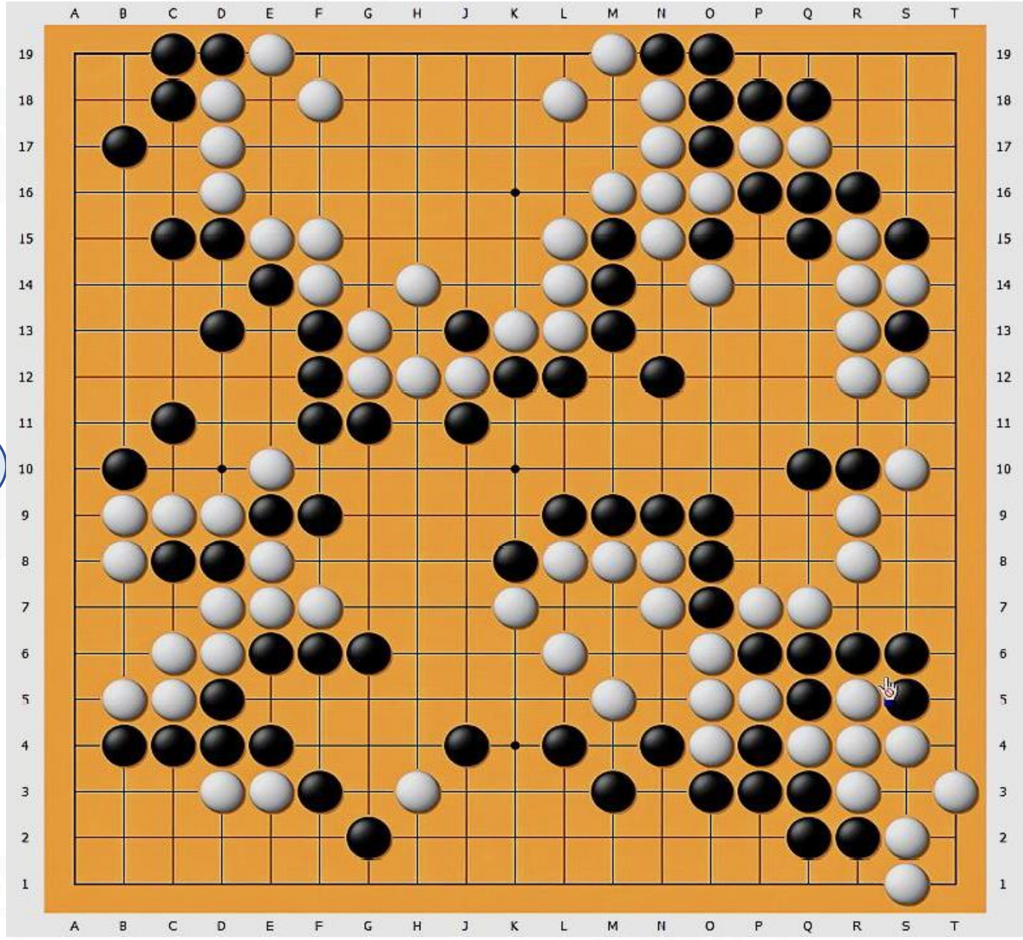
MCTS is another **automated** planning, and it behaves more **diversely**

AlphaGo

Monte Carlo Tree Search

Like playing chess, **simulate millions possible moves** in mind and choose the **“best”** step





Monte Carlo Method

Monte Carlo Method

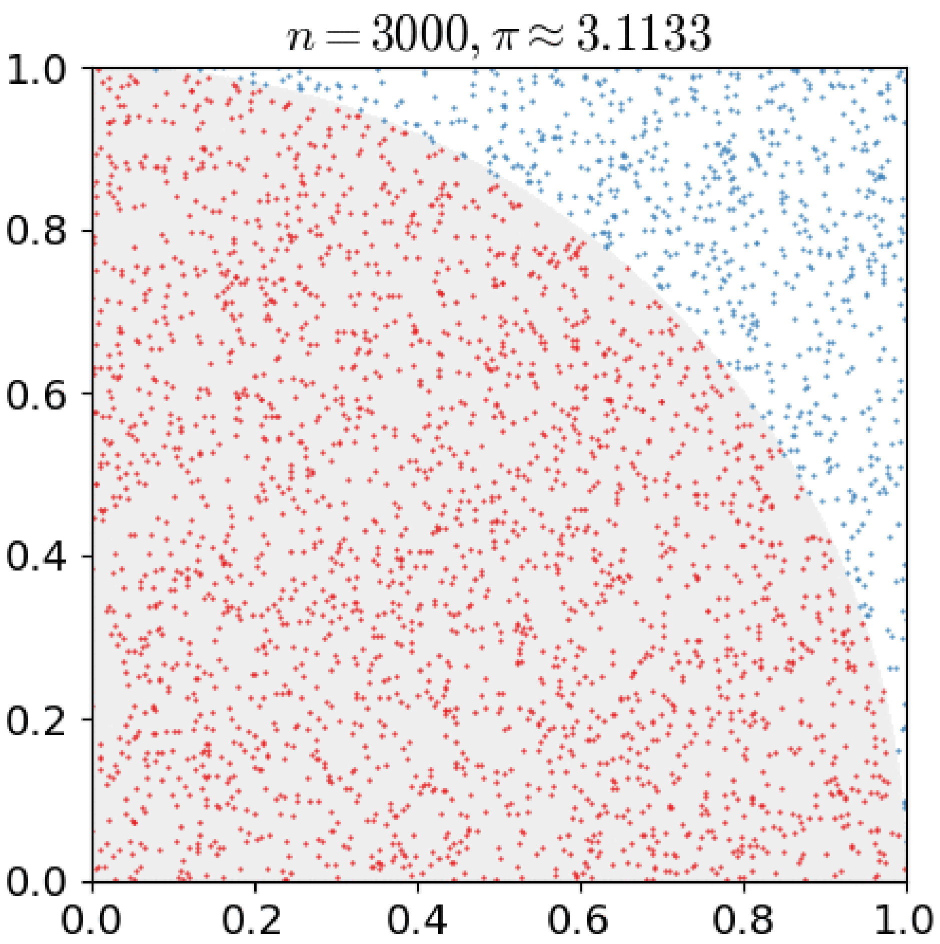
- A broad class of computational algorithms that rely on **repeated random sampling** to obtain numerical results

Monte Carlo Tree Search

Rémi Coulom

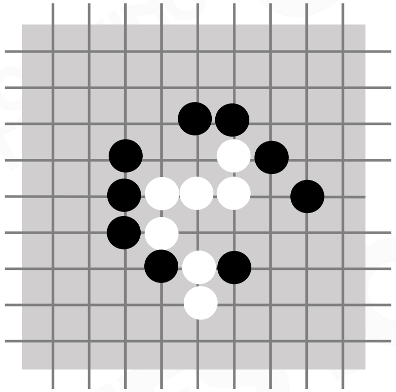
From Wikipedia, the free encyclopedia

Rémi Coulom (born 1974) is a French computer scientist,[1][2] once an assistant professor of computer science at the Lille 3 University, and the developer of Crazy Stone, a computer Go program.[3]

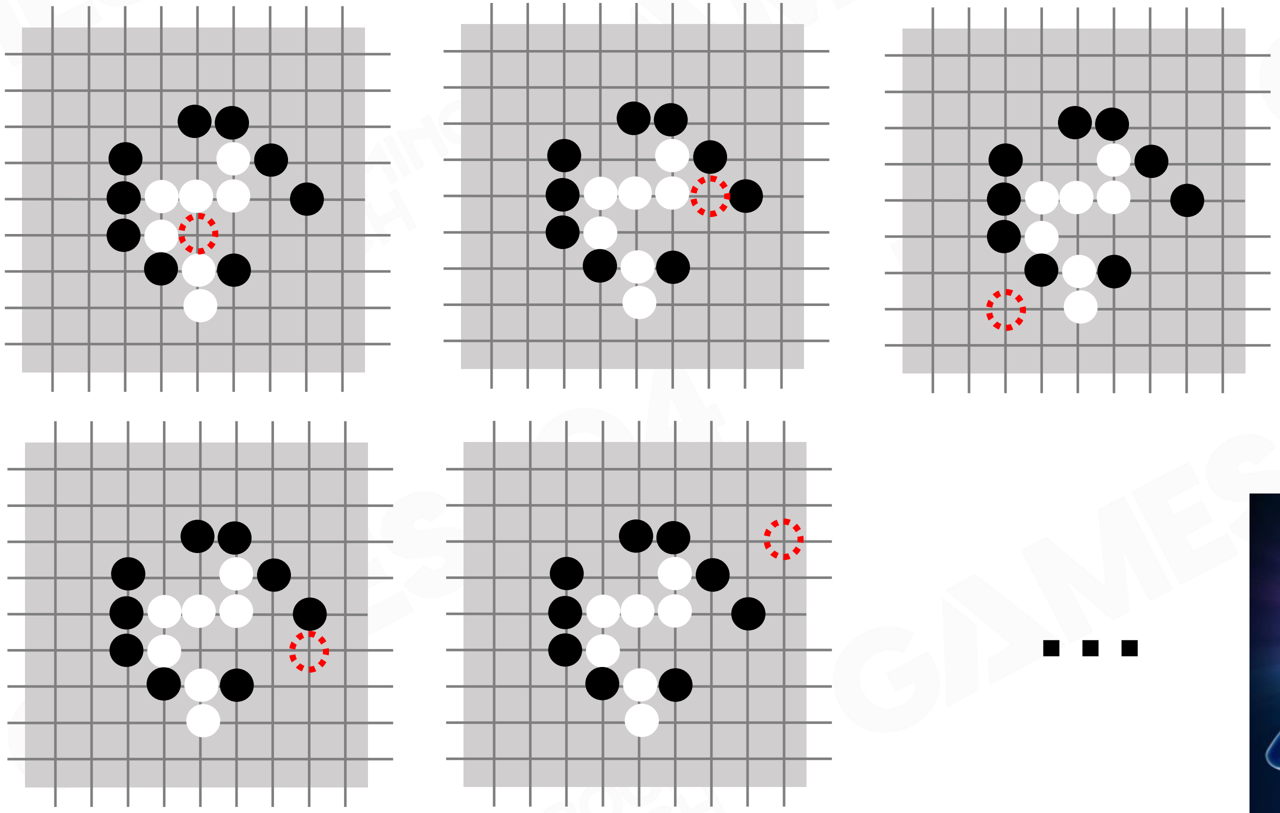


A typical Monte Carlo method to calculate π

Monte Carlo Tree Search



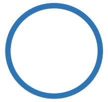
Current State



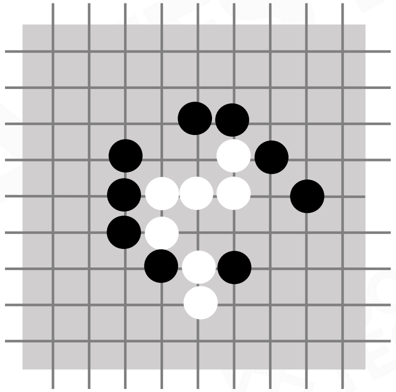
Possible Actions

States and Actions

State
- The state of game
- Represented by a node

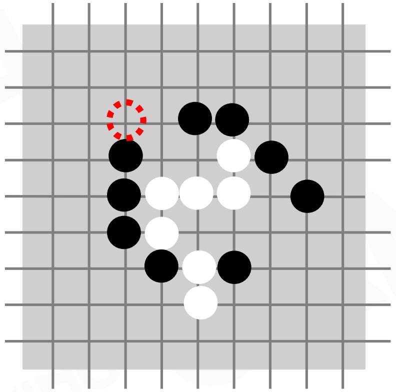


Node = State



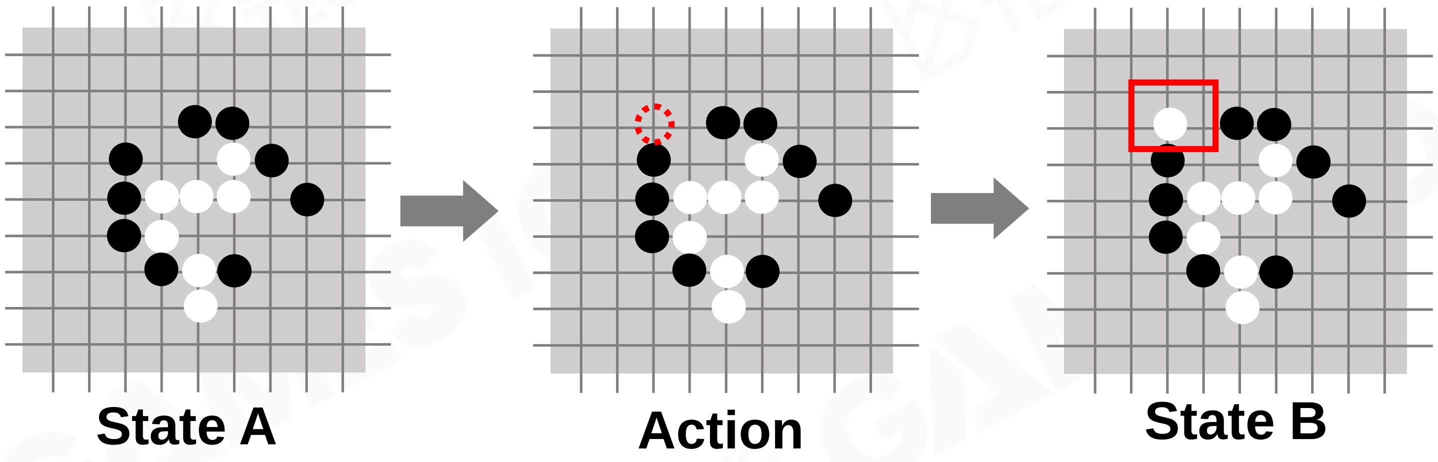
Action
- One step operation of AI
- Represented by an edge

Edge = Action



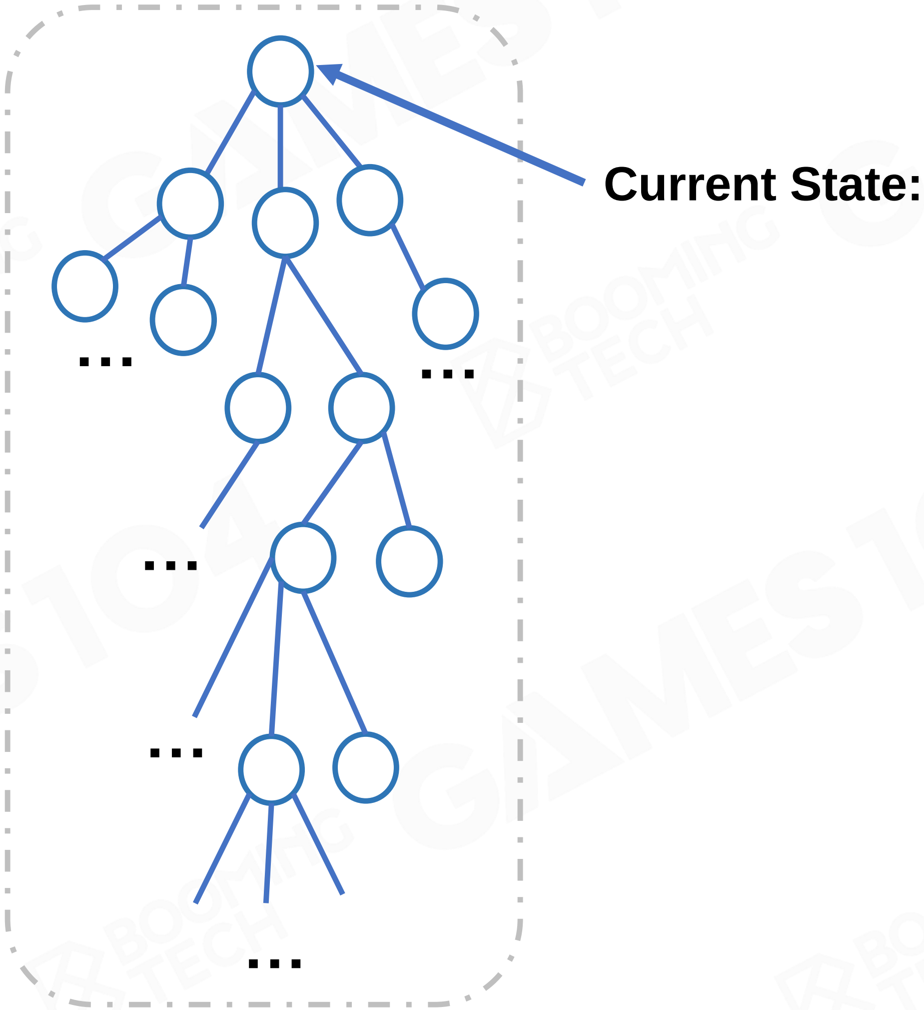
States Transfer

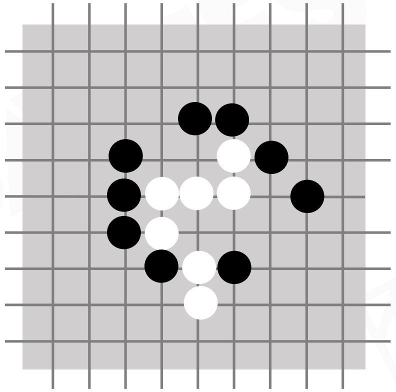
Transfer state from A to B by action



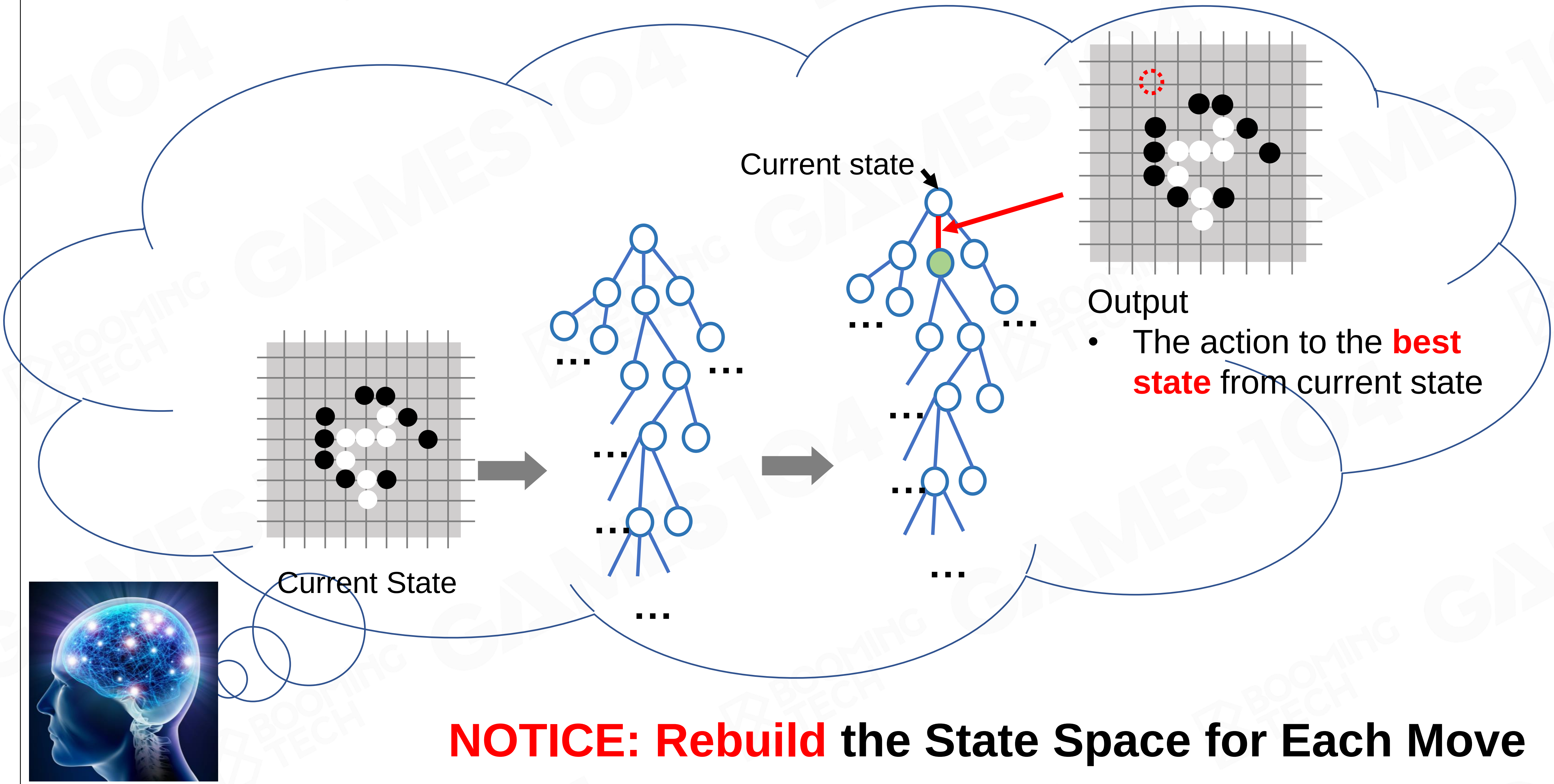
State Space

A Tree Structured State Space :
The set of states that can be reached from the **current state** after a possible sequence of actions





State Space



NOTICE: Rebuild the State Space for Each Move

Simulation : Playing a Game in Mind Quickly

Simulation

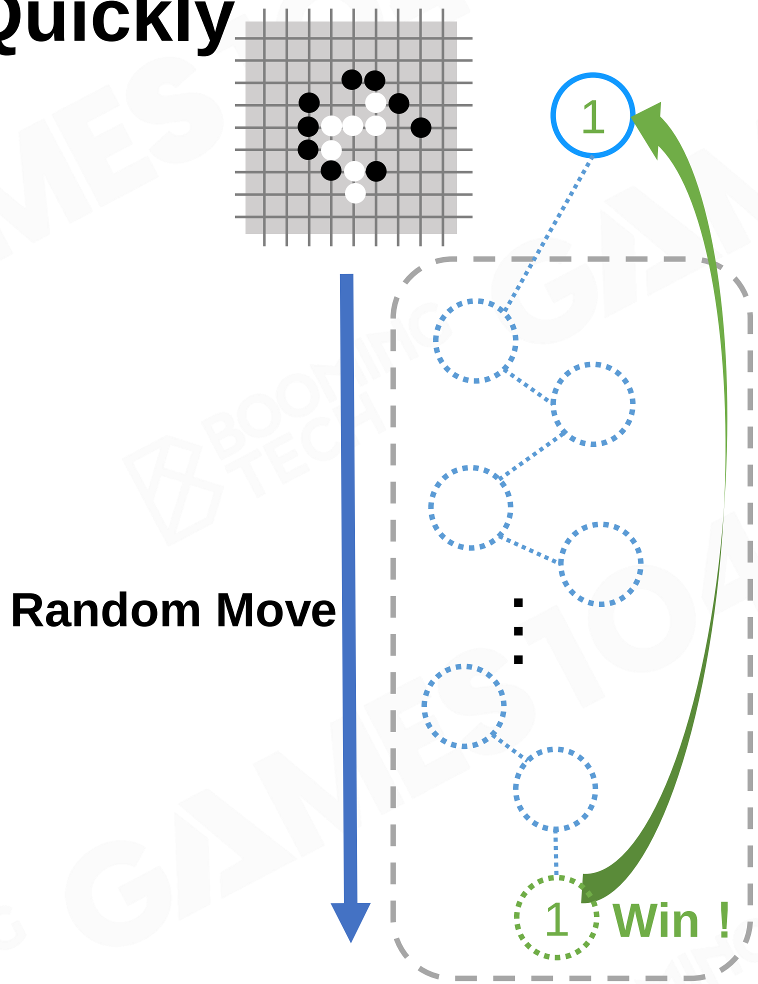
- Run from the state node according to the **Default Policy** to produce an **outcome**

In the case of Go

- Apply random moves from the state until the game is over
- Return 1 (win) or 0 (loss) depending on the result

Default Policy

- A meaningful but quick rule or neural network to play the game



Simulation of Go

How to evaluate the states?

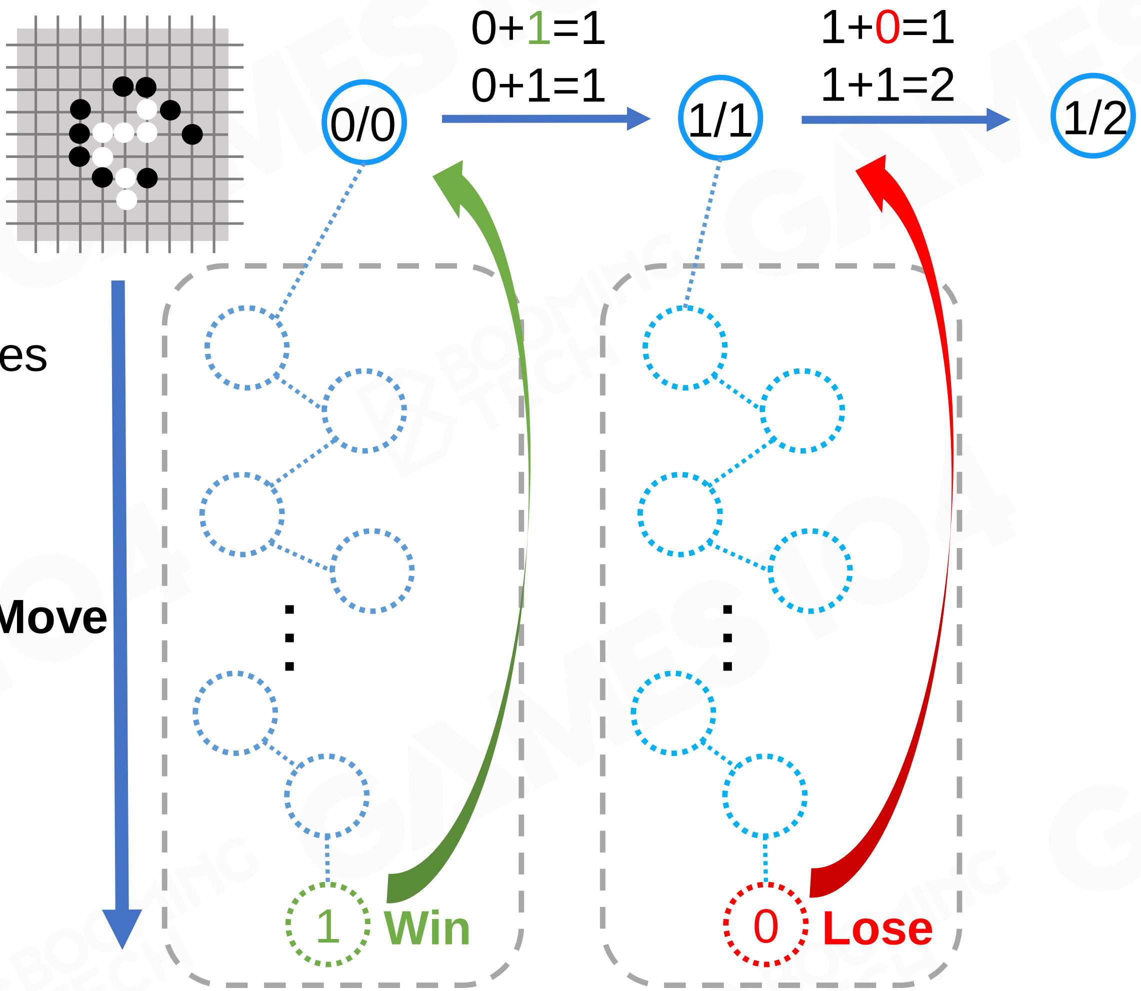
Evaluation Factors

- Q: **Accumulation** of Simulation Results
- N: Number of simulations

Maybe not direct simulation but from child nodes

Q/N

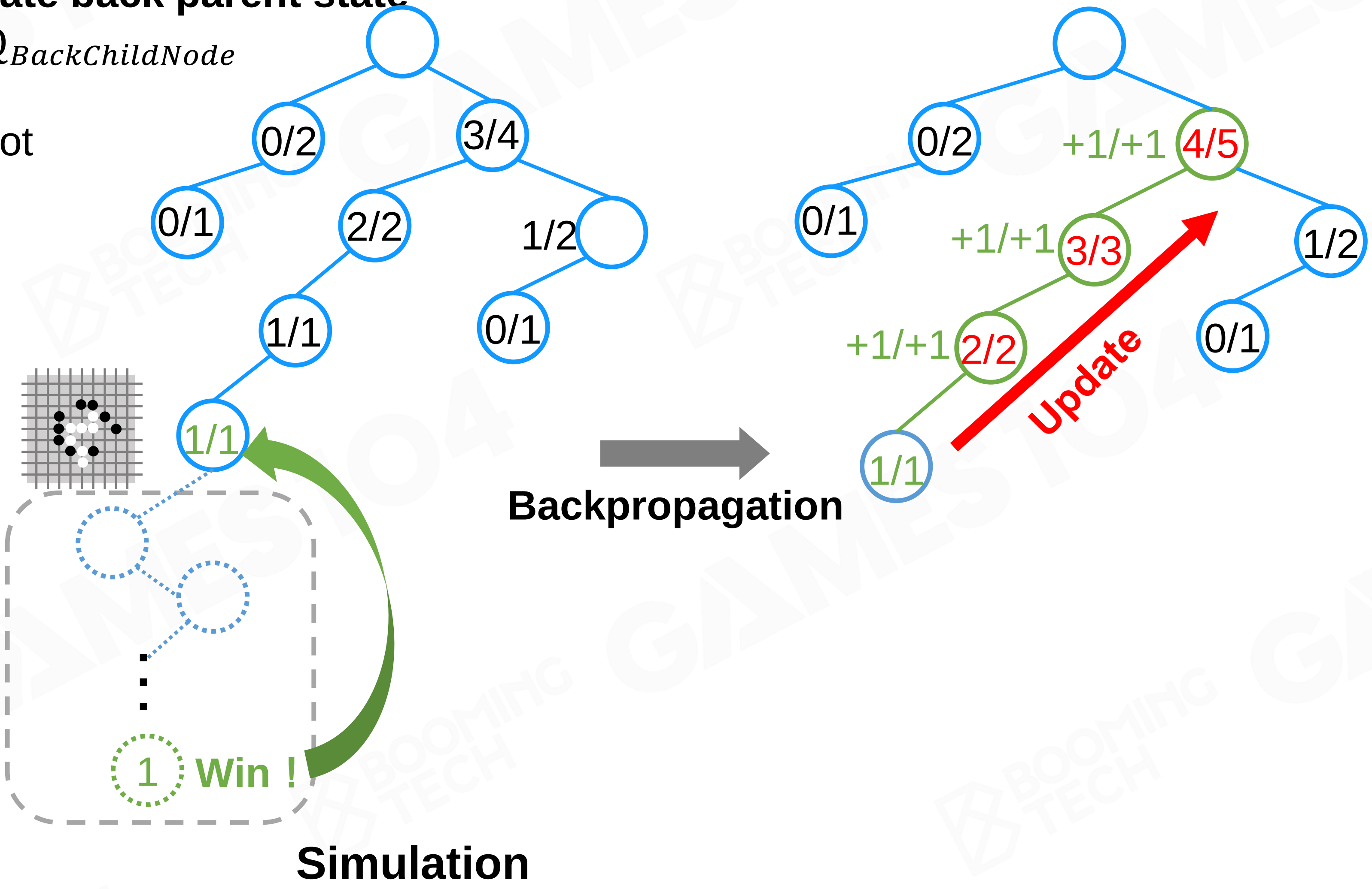
Evaluation Factors



Backpropagate

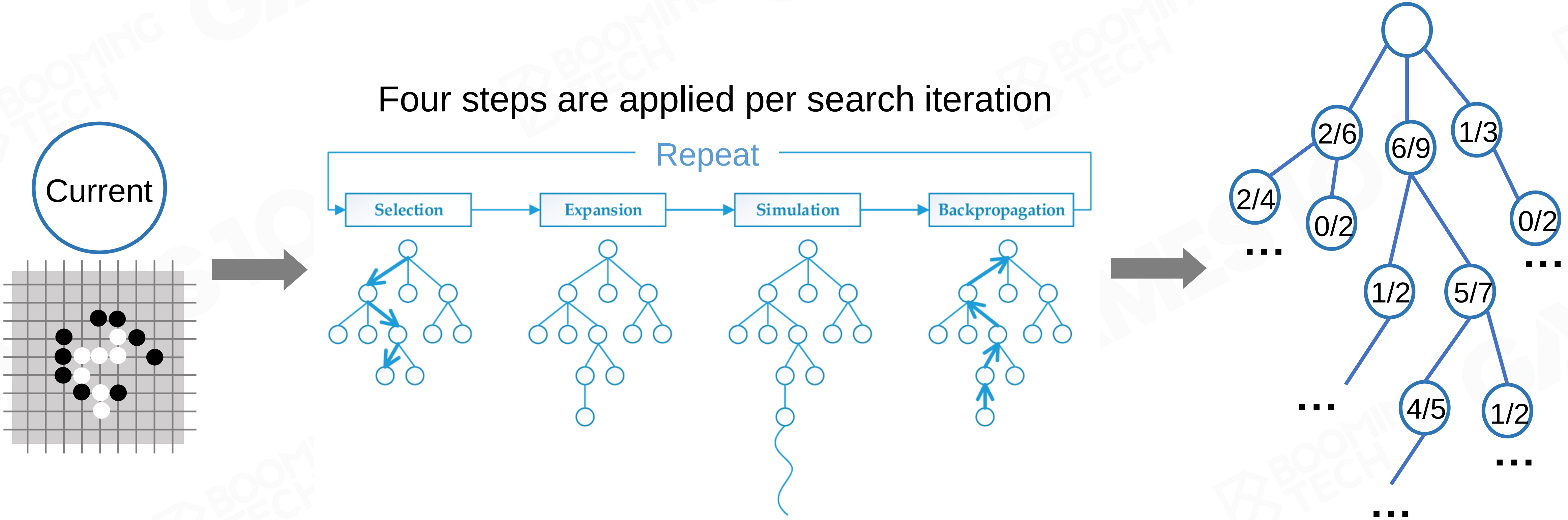
Propagate influence of child state back parent state

- $Q_{FatherNode} = Q_{FatherNode} + Q_{BackChildNode}$
- $N_{node} = N_{node} + 1$
- Repeat it until reaching the root



Iteration Steps

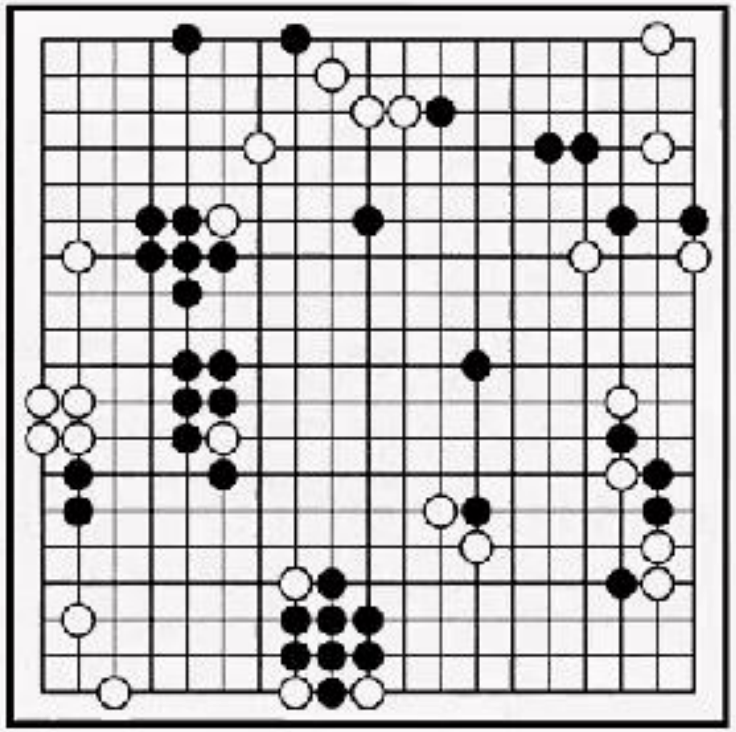
Selection : select the most urgent “**expandable**” node
Expansion : expand the tree by selecting an **action**
Simulation : simulate from the new node and produce an **outcome**
Backpropagate : backpropagate the **outcome** of simulation from the new node



Search in "Infinite" State Space

Generally impossible to traverse the state space

- We prioritize exploring **the most promising regions** in state space
- Pre-set a **computational budget** and stop exploring the state space when the budget is reached

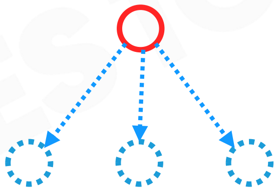


Selection — Expandable Node

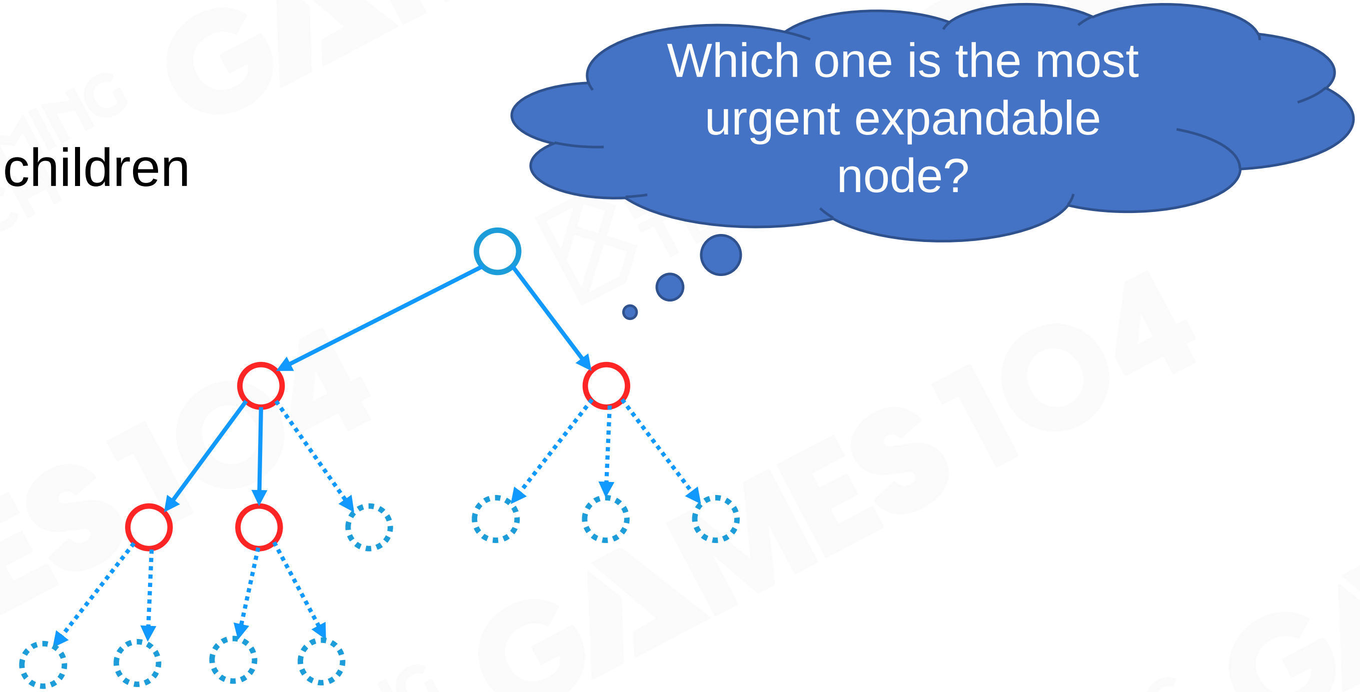
Select the **most urgent** “**expandable**” node

“expandable” node

- Nonterminal state and has unvisited children
- Example:



The Current State Before First Iteration



The State has unvisited children

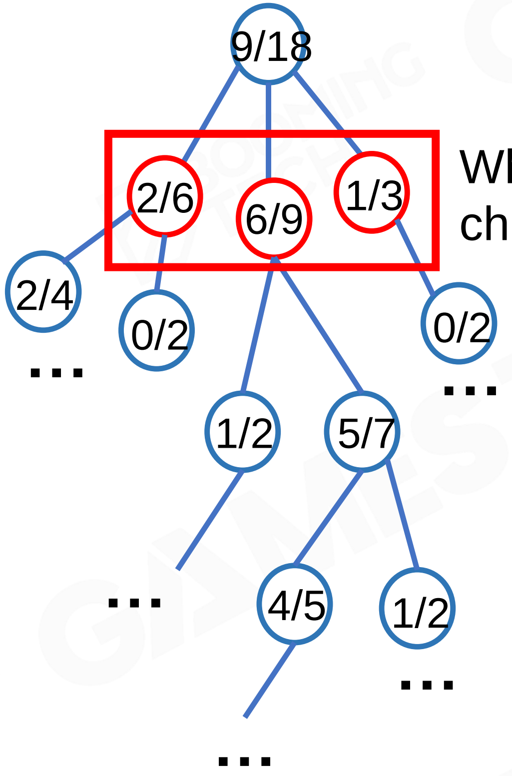
Selection — Exploitation and Exploration

Exploitation

- Look in areas which appear to be **promising**
- Select the child which has **high Q/N value**

Exploration

- Look in areas that have **not been well sampled** yet
- Select the child which has **low number of visits**

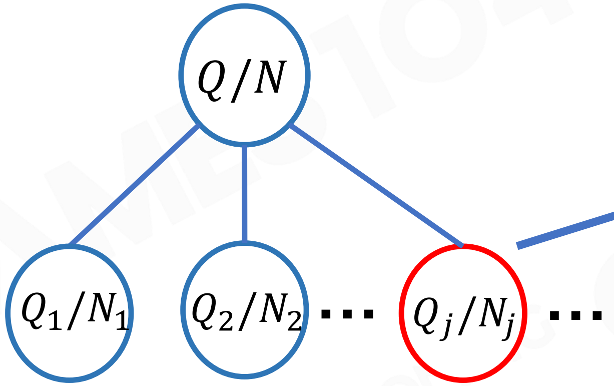


What is **most promising** child node ?

UCB (Upper Confidence Bounds)

How to balance exploration and exploitation?

- Use UCB (Upper Confidence Bounds) formula
  - UCB_j : the UCB value of the node j
  - Q_j : the total reward of all playouts that passed through node j
  - N_j : the number of times node j has been visited
  - N : the number of times the parent node of node j has been visited
  - C : a constant, adjusted to lower or increase the amount of exploration performe

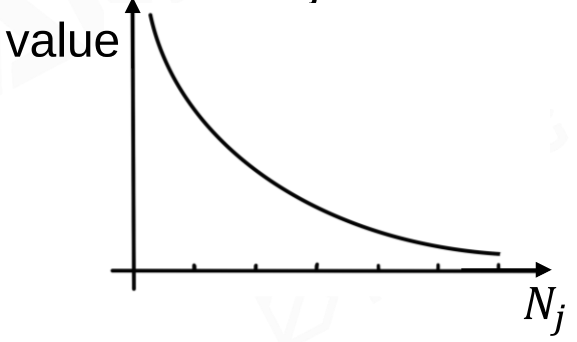


$$UCB_j = \boxed{\frac{Q_j}{N_j}} + C \cdot \boxed{\sqrt{\frac{2\ln(N)}{N_j}}}$$

Exploitation

Exploration

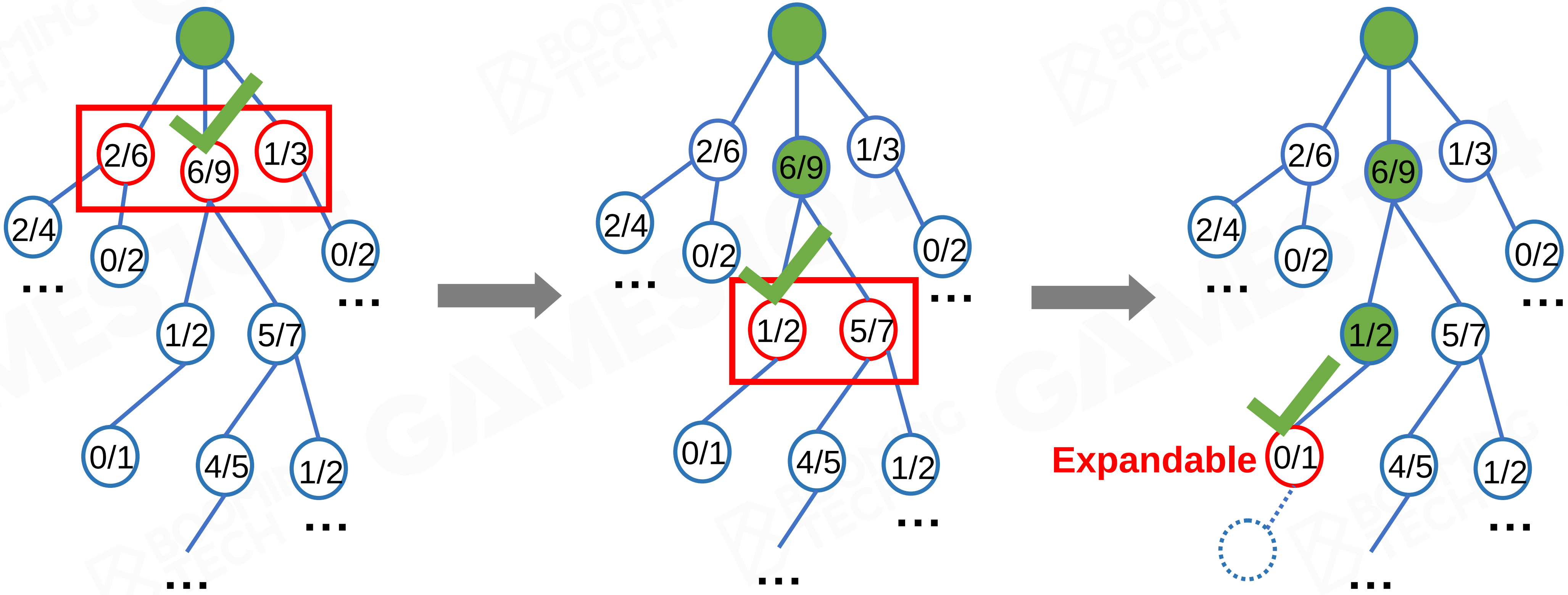
The smaller the N_j, the larger the value



Selection

How to select the most urgent expandable node

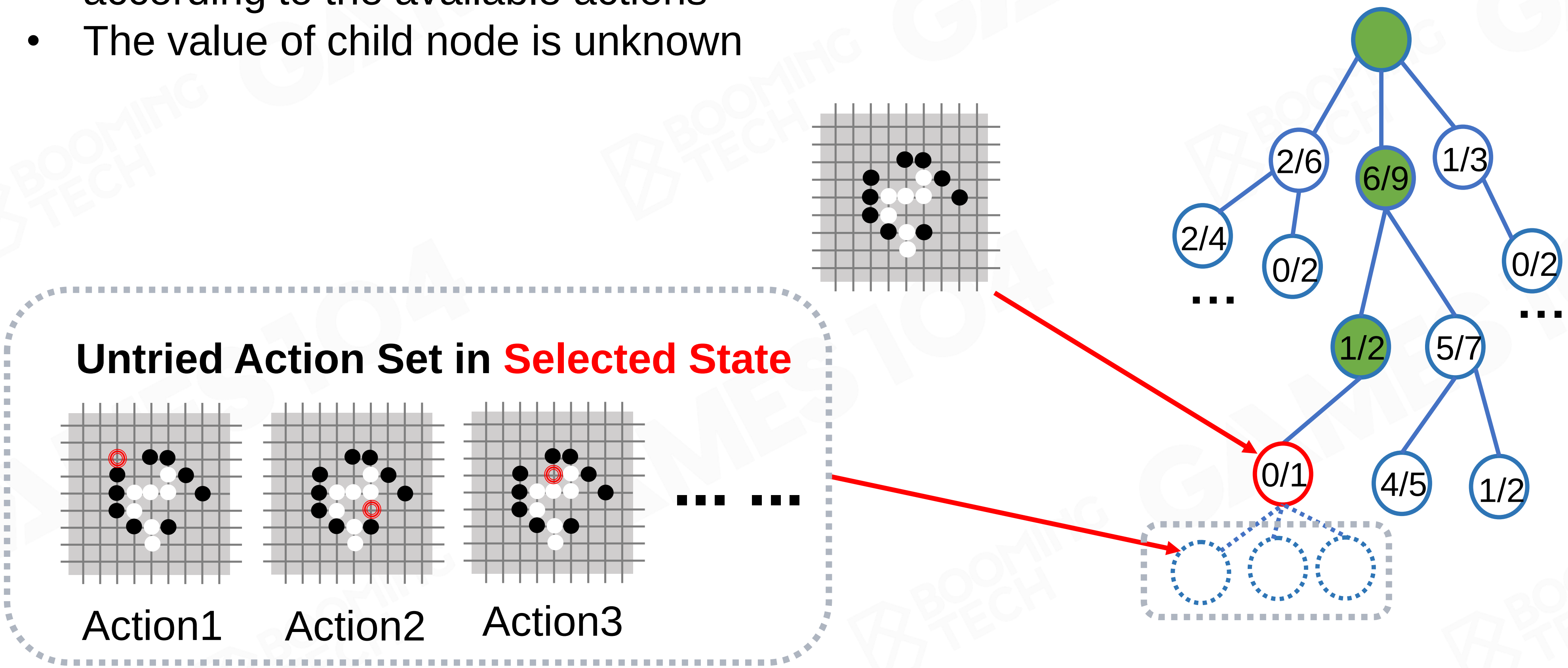
- Always Search from the root node
- Find the highest UCB value child node (promising child) of current node
- Set promising child as current node
- Iterate above steps until current node is expandable. Set current node as selected node



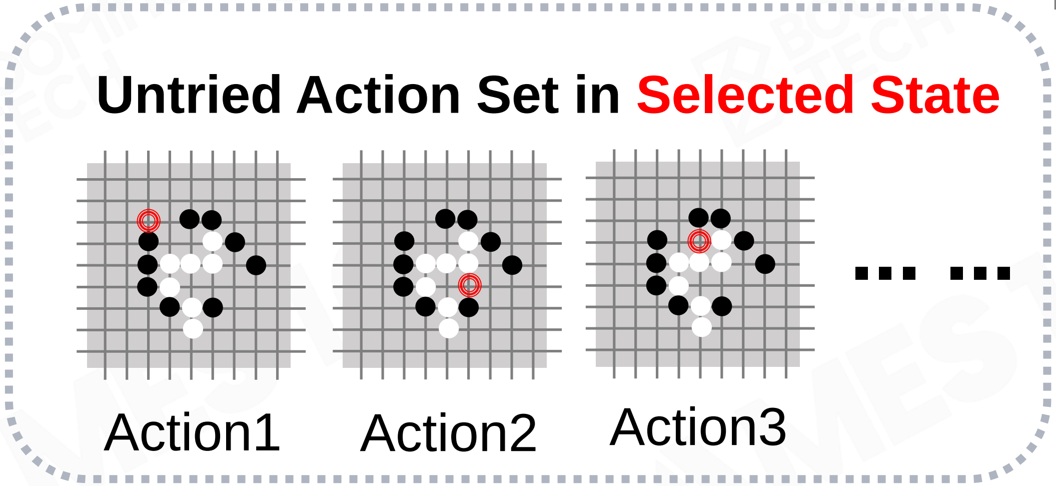
Expansion

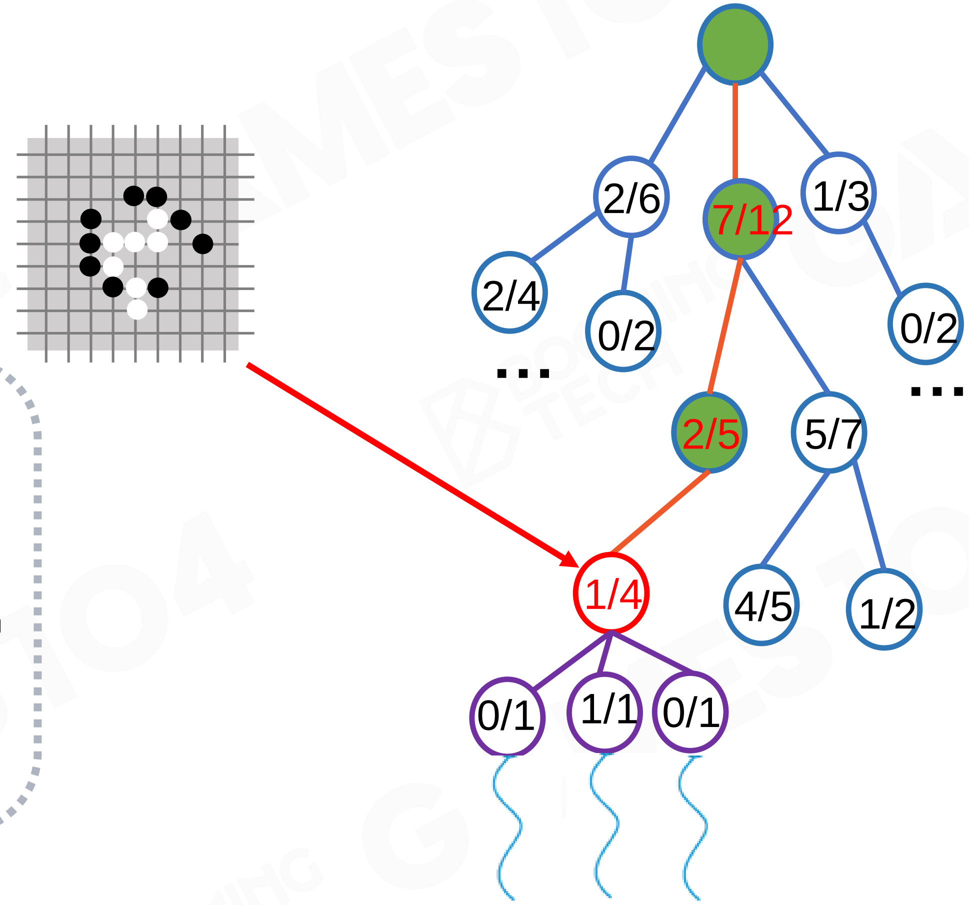
Expansion

- One or more new child nodes are added to selected node, according to the available actions
- The value of child node is unknown



Simulation and Backprogation

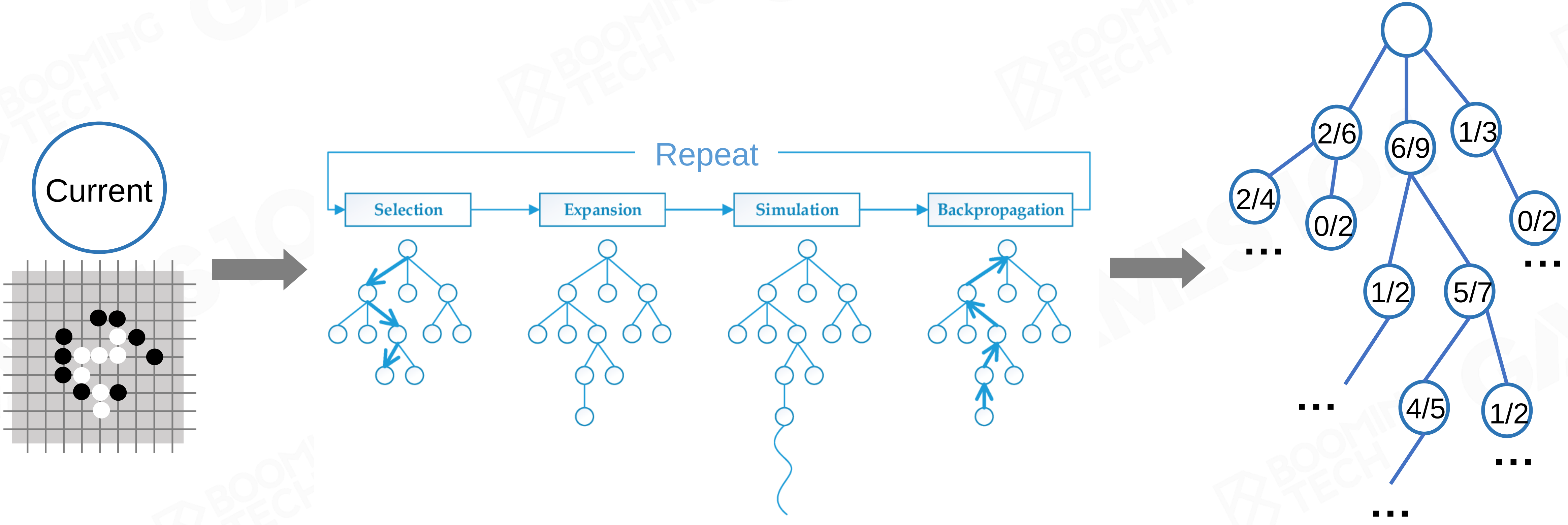




The End Condition

Computational budget

- Memory size (the number of nodes)
- Computation time

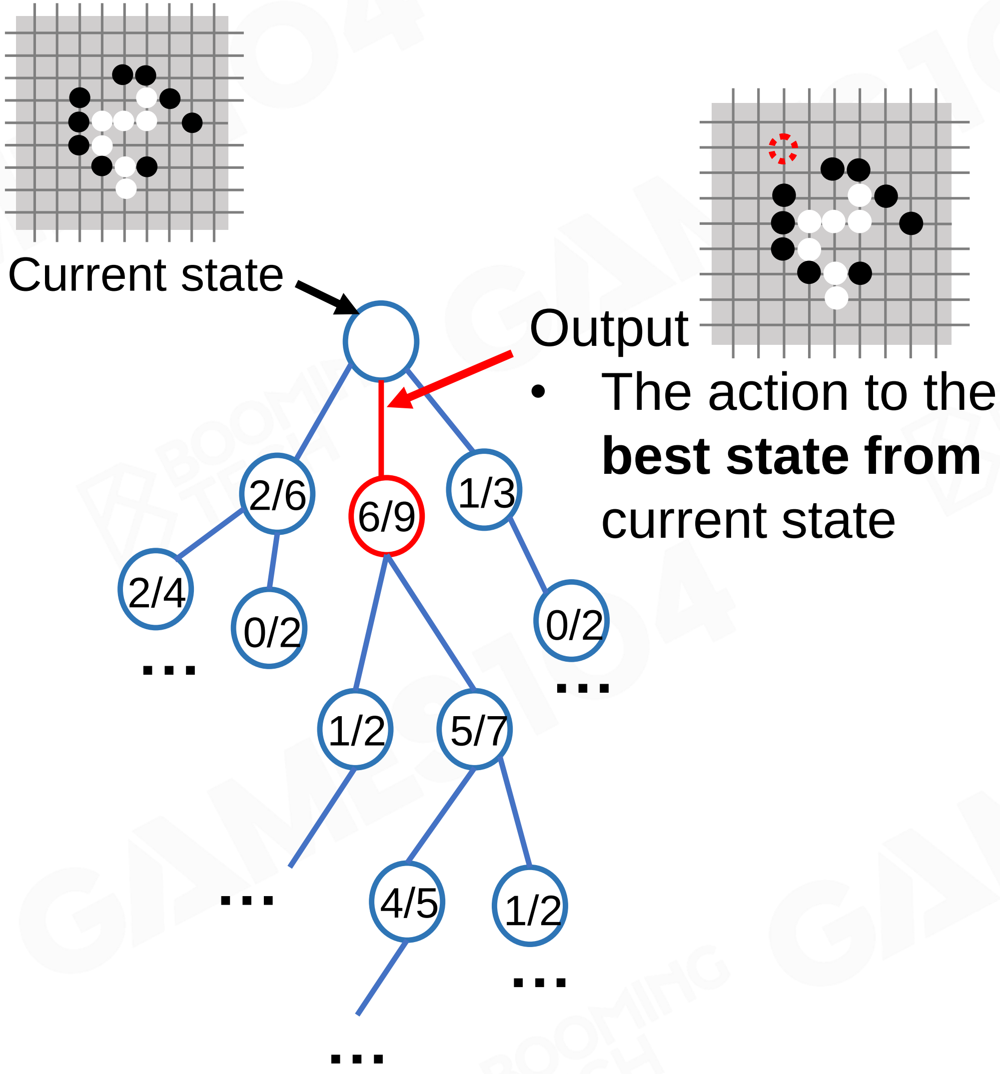


How to Choose the Best Move ?

The “best” child node of current state node

- **Max child**: Select the root child with the **highest Q-value**
- **Robust child**: Select the **most visited** root child
- **Max-Robust child**: Select the root child with **both** the highest visit count and the highest reward. If none exist, then continue searching until an acceptable visit count is achieved
- **Secure child**: Select the child which **maximises** a lower confidence bound (LCB)

$$LCB_j = \frac{Q_j}{N_j} - C \cdot \sqrt{\frac{2\ln(N)}{N_j}}$$



- The action to the **best state from** current state

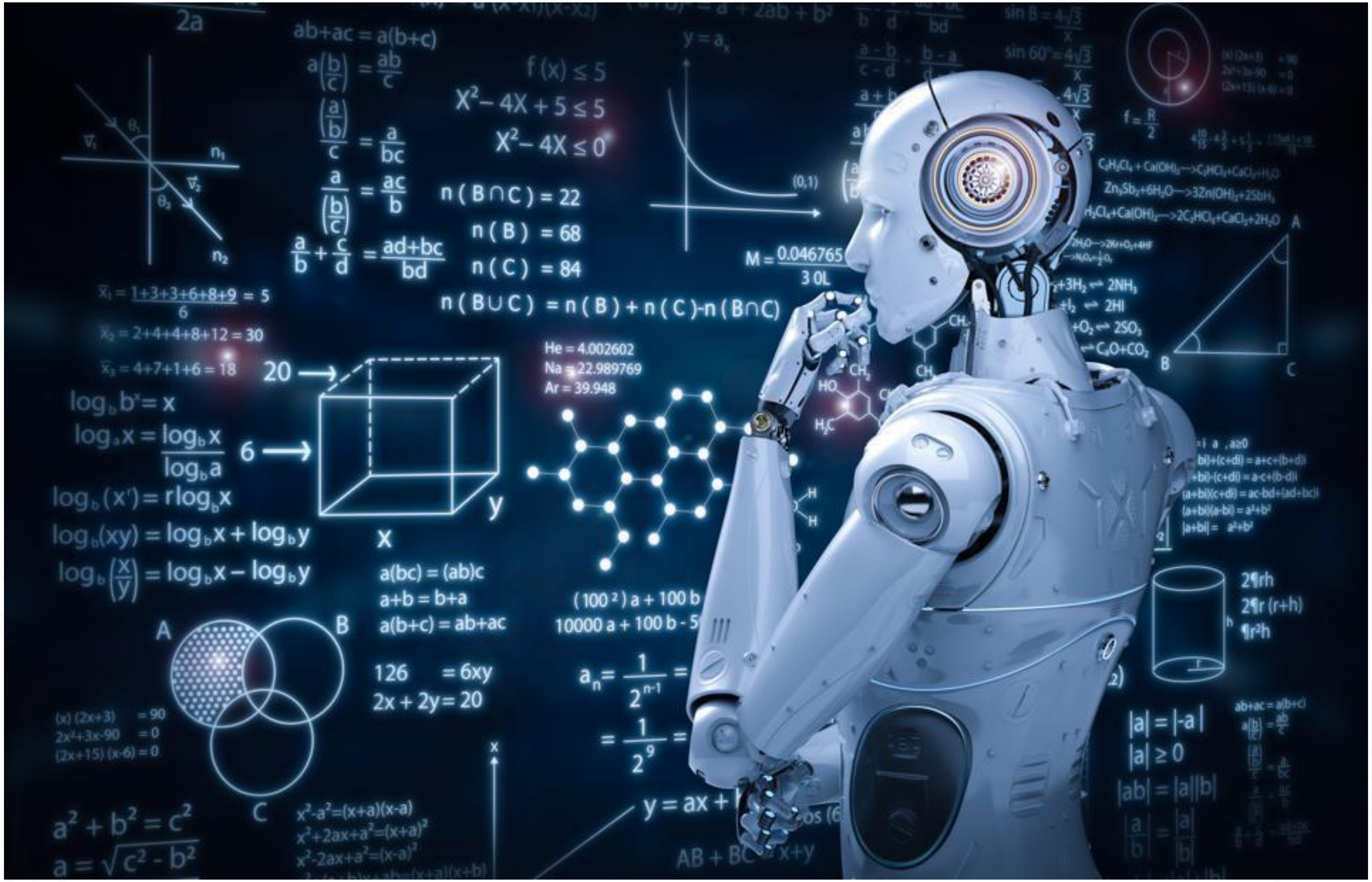
Conclusion

Pros:
- MCTS agent behaves diverse
- Agent makes the decision totally by itself
- Can solve the problem of large search space

Cons:
- The action and state are hard to design for most real-time games
- It is hard to model for most real-time games

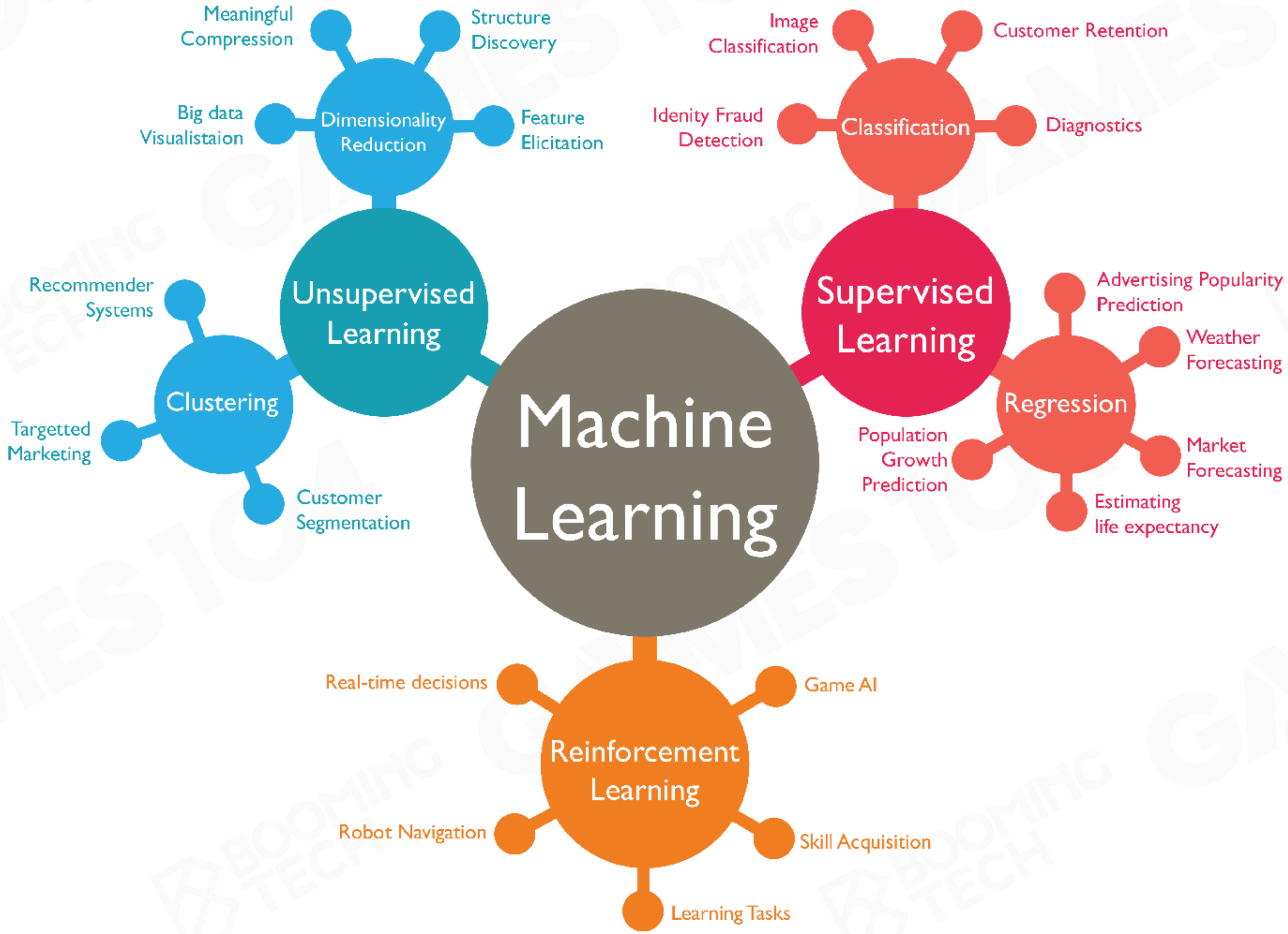
Machine Learning Basic

Machine Learning



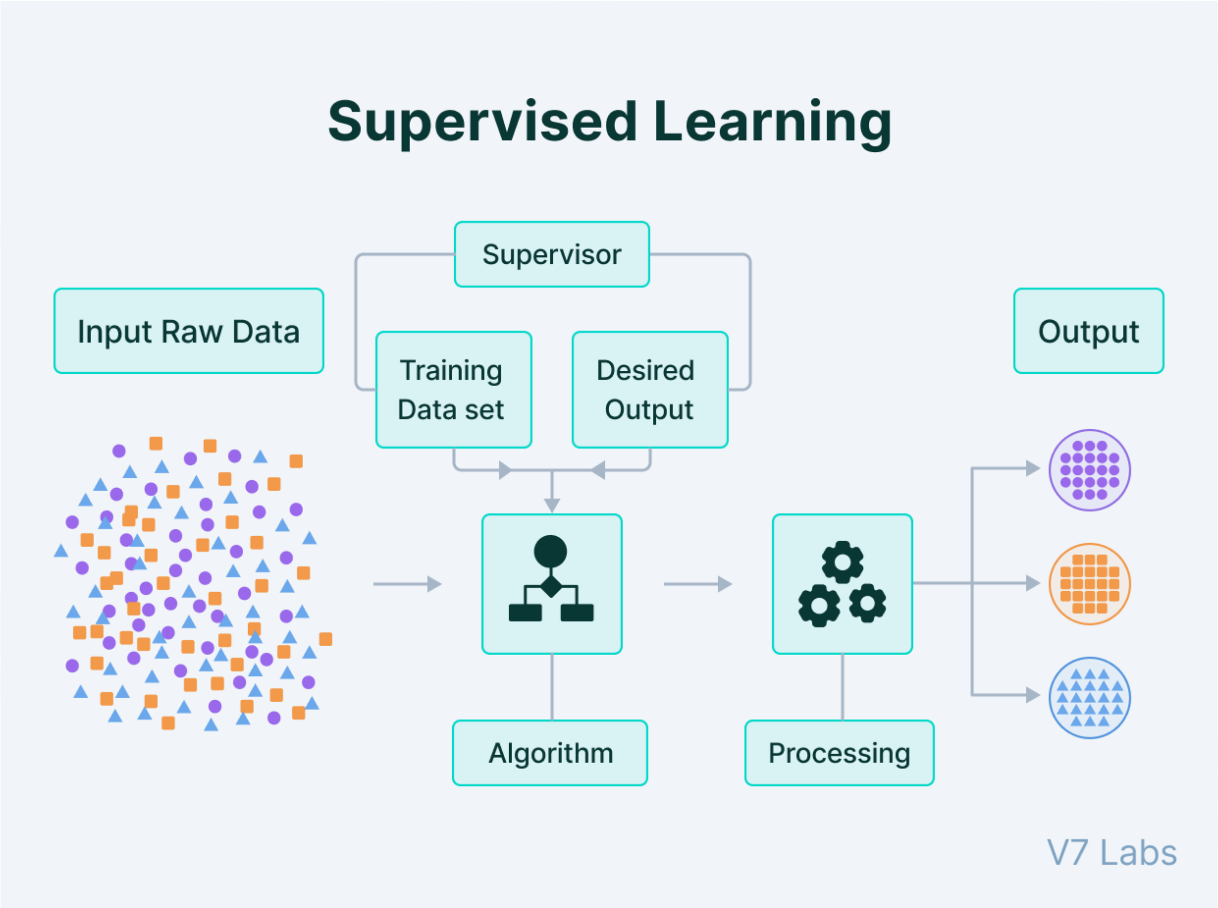
Four Types of Machine Learning

- Supervised learning
- Unsupervised learning
- Semi-supervised learning
- Reinforcement learning



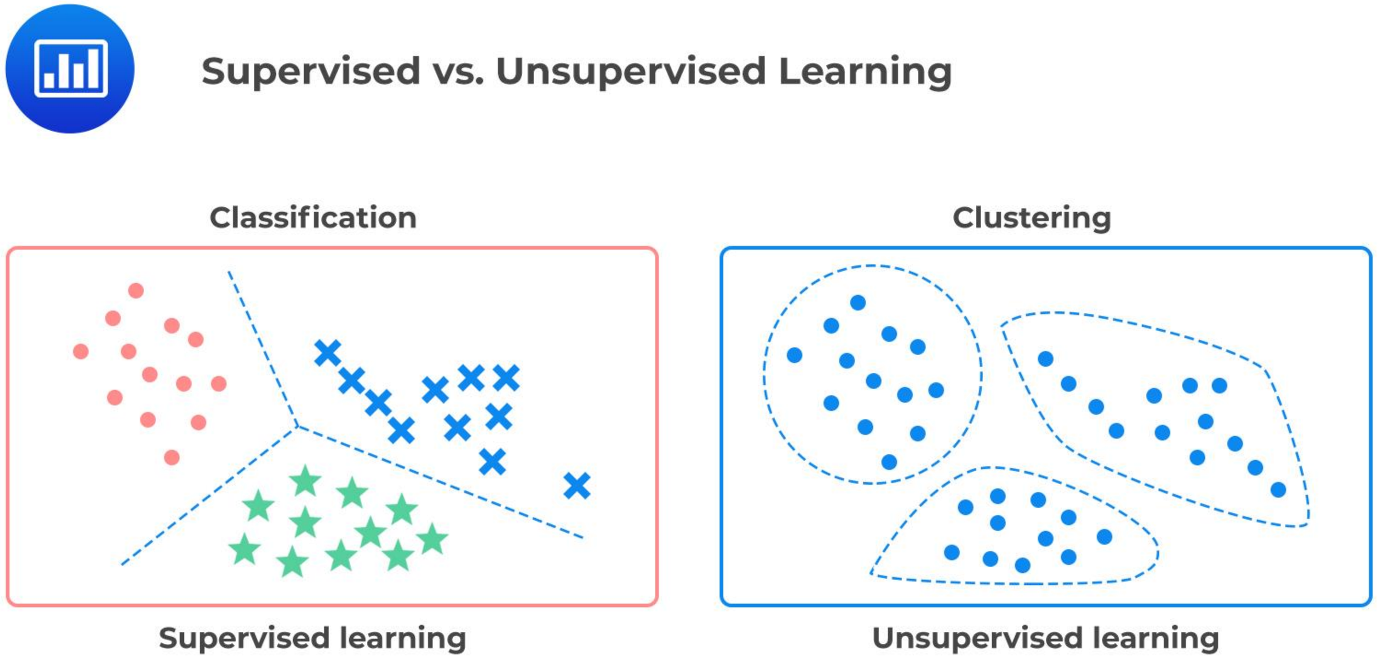
ML Types: Supervised Learning

- Learn from labeled data



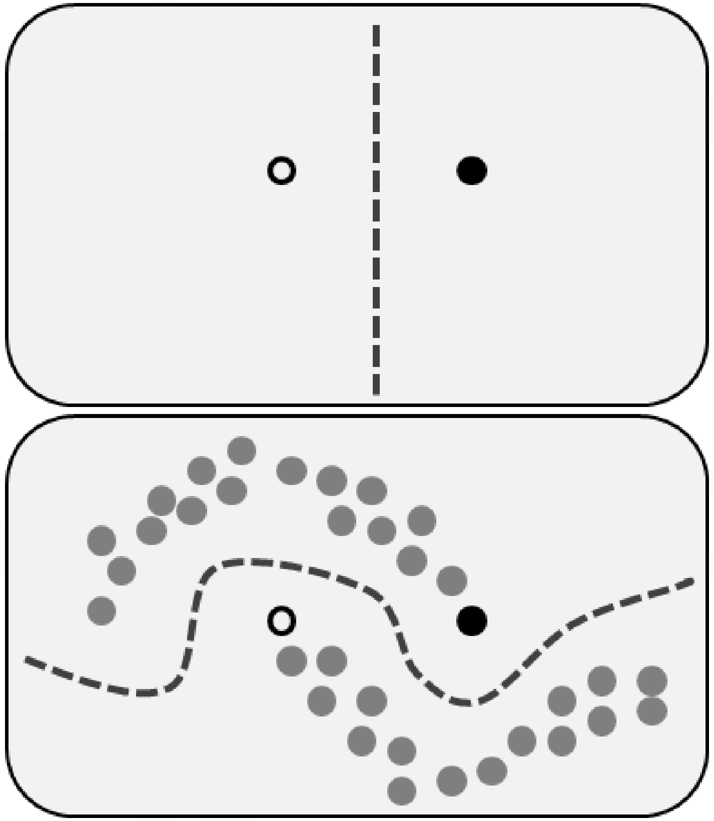
ML Types: Unsupervised Learning

- Learn from unlabeled data



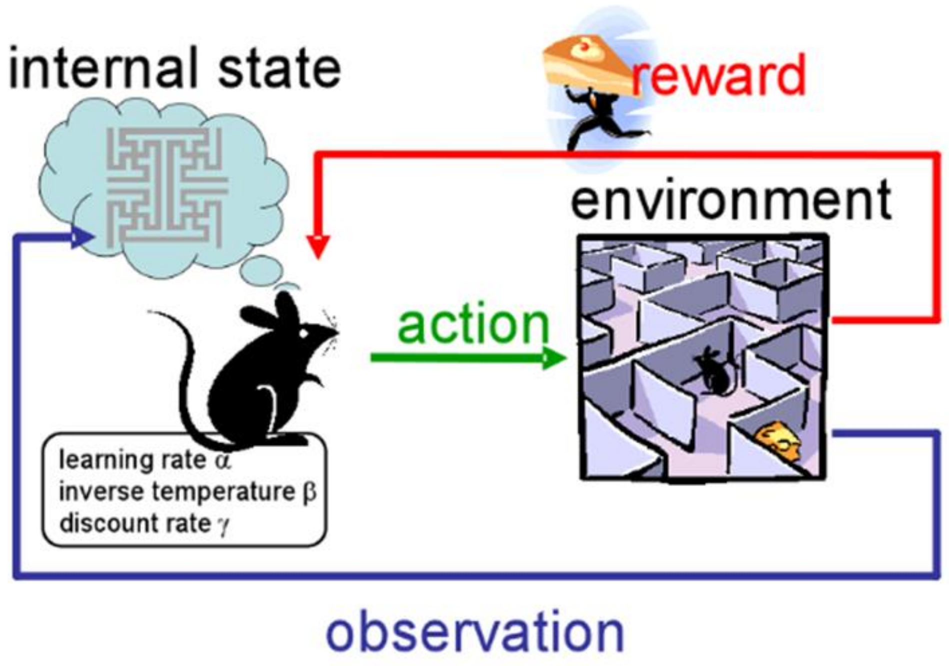
ML Types: Semi-supervised Learning

- Learn from a lot of unlabeled data and very scarce labeled data.



ML Types: Reinforcement learning

- Learn from an interaction process with environment

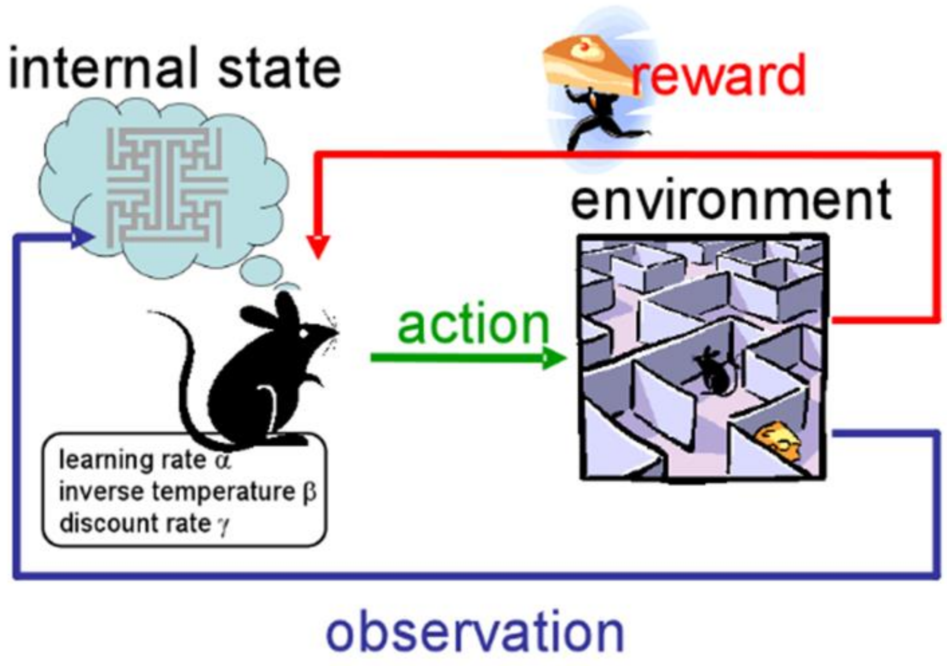


Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward

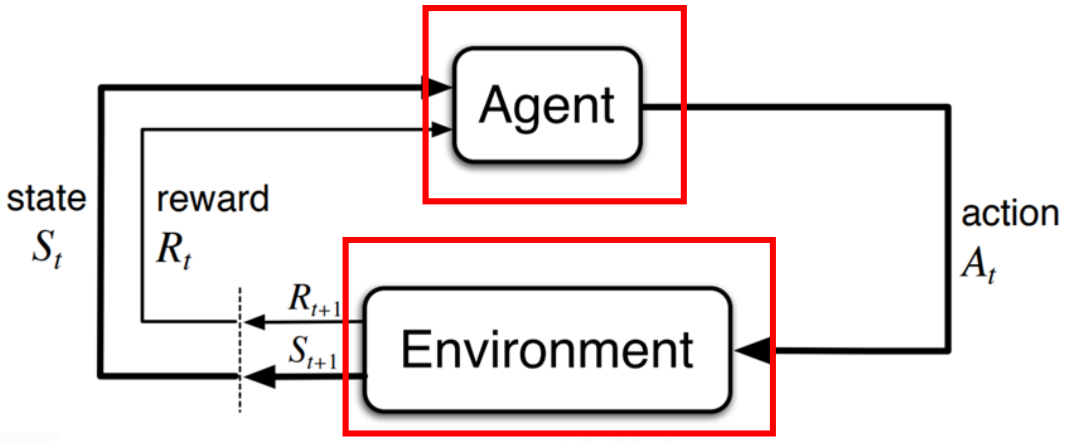
- **Trial-and-error search**
- The learner must discover which actions yield the most reward by trying them

- **Delayed reward**
- Actions may affect the immediate reward, the next situation and all subsequent rewards



Markov Decision Process - Basic Elements (1/4)

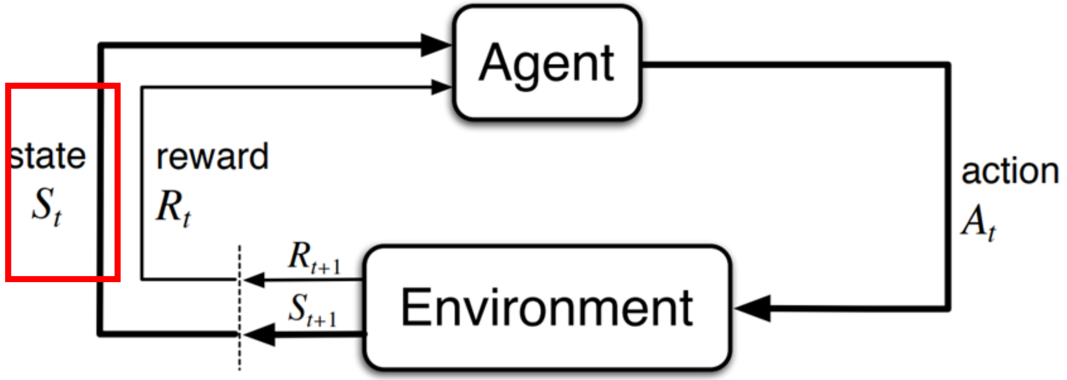
- Agent
  The learner and decision maker
- Environment
  The thing the agent interacts with, comprising everything outside the agent



Markov Decision Process - State (2/4)

State is the observation of the agent, and the data structure is designed by human

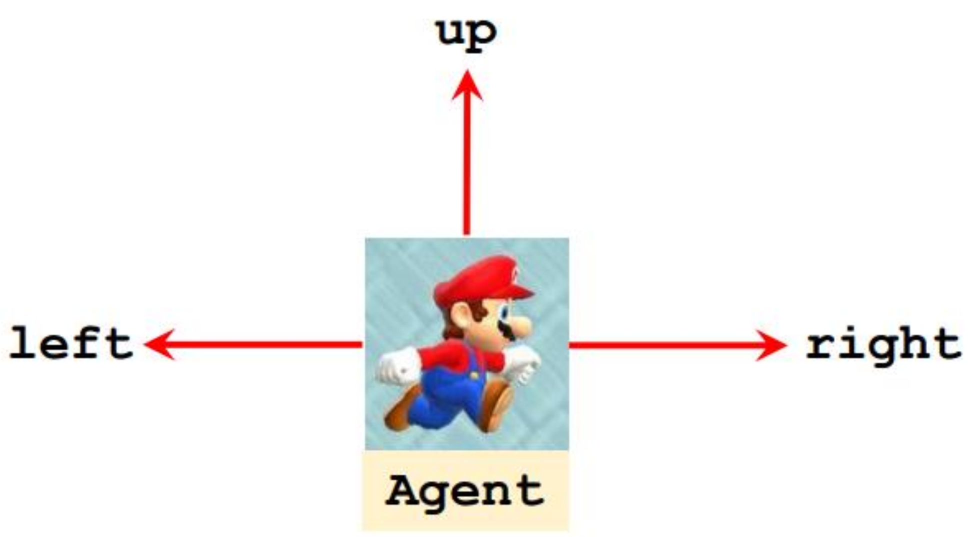
State ***s*** (this frame)

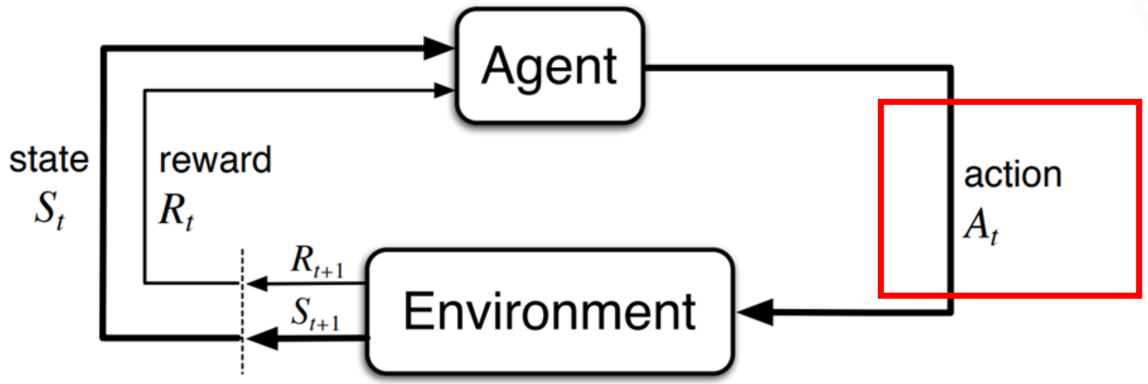


Markov Decision Process - Action (3/4)

Action is the minimal element the agent could behave in the game
It is also designed by human

Action $a \in$ {left, right, up}



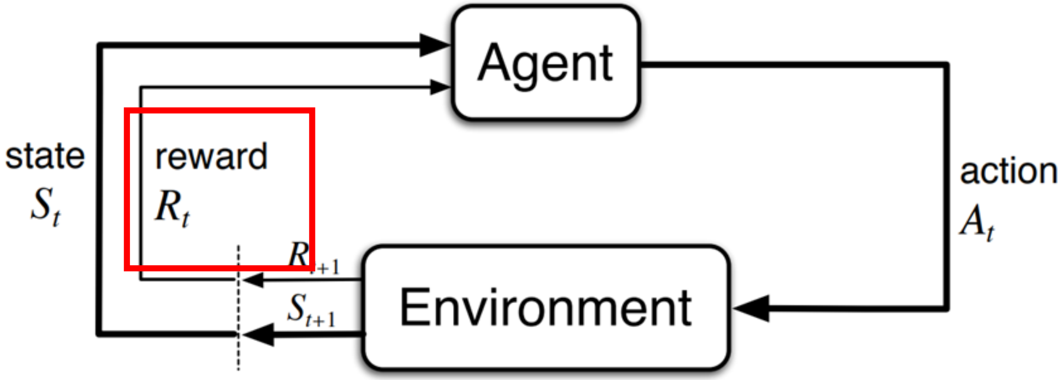


Markov Decision Process - Reward (4/4)

A special signal the agent receives at each time step passing from environment to the agent

Reward ***R***

- Collect a coin: ***R*** = +1
- Win the game: ***R*** = +10000
- Touch a Goomba: ***R***= -10000 (game over)
- Nothing happens: ***R*** = 0



MDP Mathematical Model

- Probability of transition
  The probability of transition from ***s*** to ***s'*** after taking action ***a***

$$p(s'|s,a) = P(S_t = s'|S_{t-1} = s, A_{t-1} = a)$$

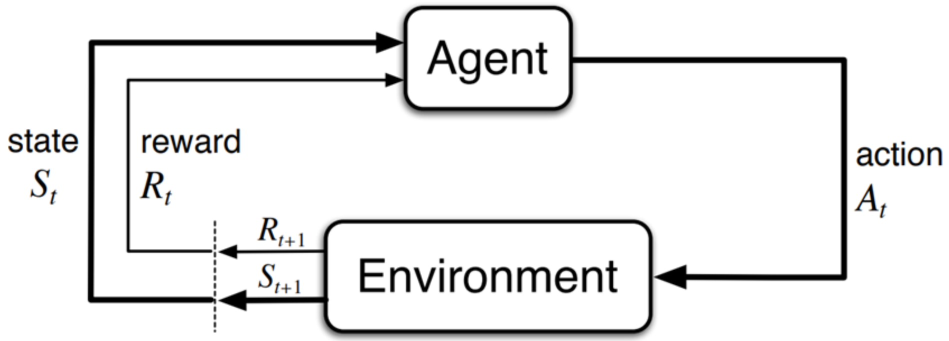
- **Policy**
  A mapping from states to probabilities of selecting each possible action

$$\pi(a|s) = P(A_t = a|S_t = s)$$

- Total reward
  The cumulative reward it receives in the long run

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$



Policy

A mapping from states to probabilities of selecting each possible action

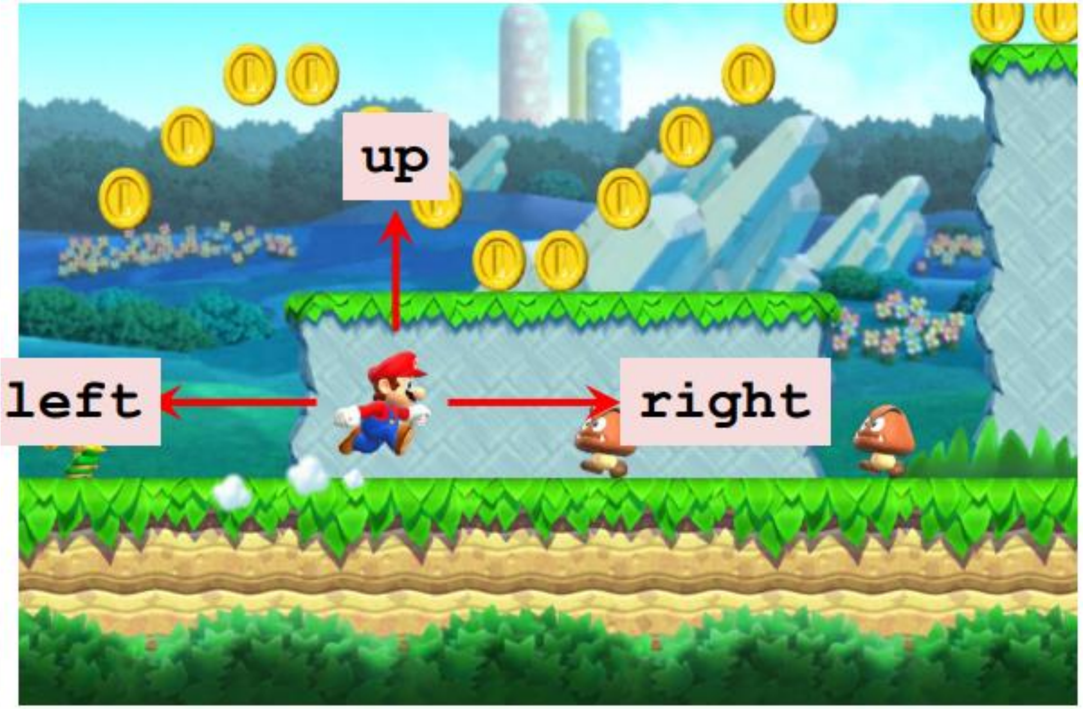
$$\pi(a|s) = P(A = a|S = s)$$

Policy π

- $\pi(a|s)$ is the probability of taking action $A = a$ given state s
- Upon observing state $S = s$, the agent's action A can be random
- For example: $\pi(\text{left}|\text{s}) = 0.2$

  $\pi(\text{right}|\text{s}) = 0.1$

  $\pi(\text{up}|\text{s}) = 0.7$



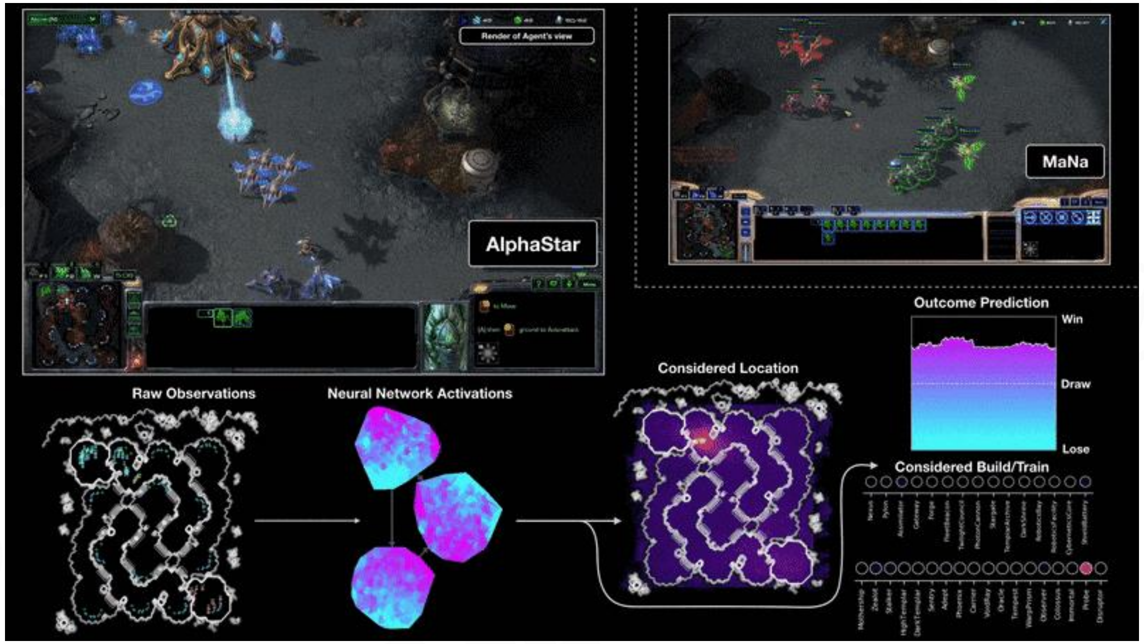
Build Advanced Game AI

Why Game AI needs Machine Learning

It is notable that all previous methods actually need human knowledge to design (include the cost of GOAP)

But players always expect AI to be able to both deal with **complicated game world** and behave **naturally and diversely**

- Traditional methods is in limited space
- Machine Learning create infinite possibilities

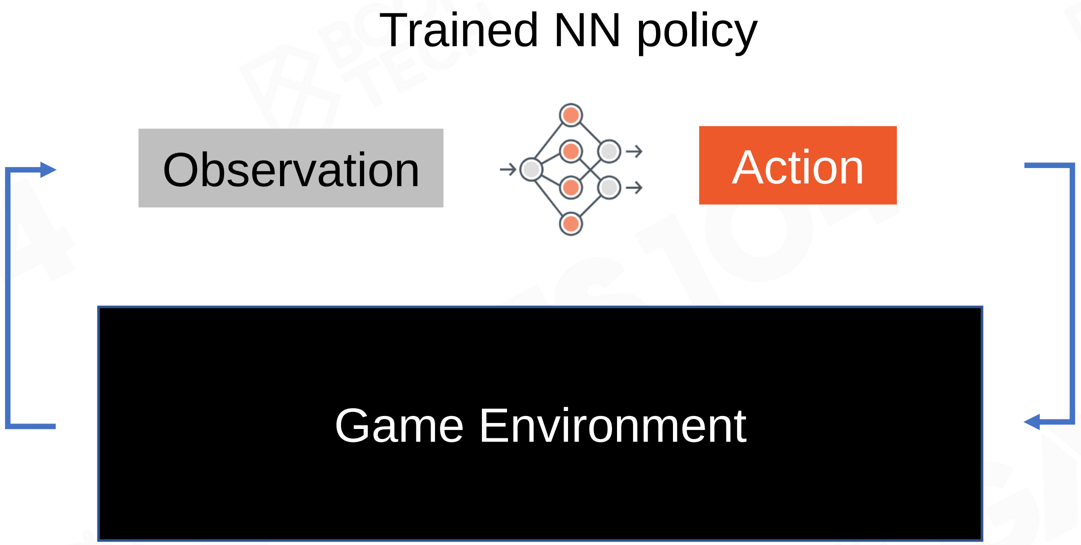


Machine Learning Framework in Game

The framework of deploying a neural network to play an agent

Observation:

- The Game State the AI could observe
  - Vector feature
    - Unit information
    - Environment information
    - Etc.
  - Image
  - …



DRL Example — Model the Game

A DRL design process should contain:

- State
- Action
- Reward
- NN design
- Training Strategy

DRL example — State

States (1/2) — Maps

Heights

Visibility: fog of war

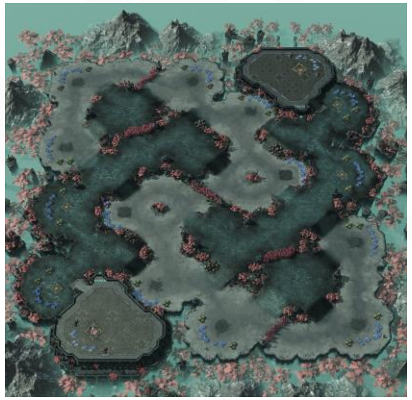
Creep

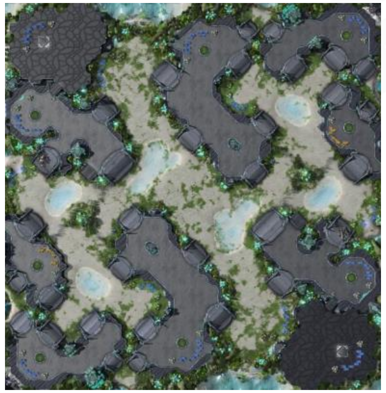
Entity owners

Alerts

Pathable

Buildable





States (2/2) — Units Information

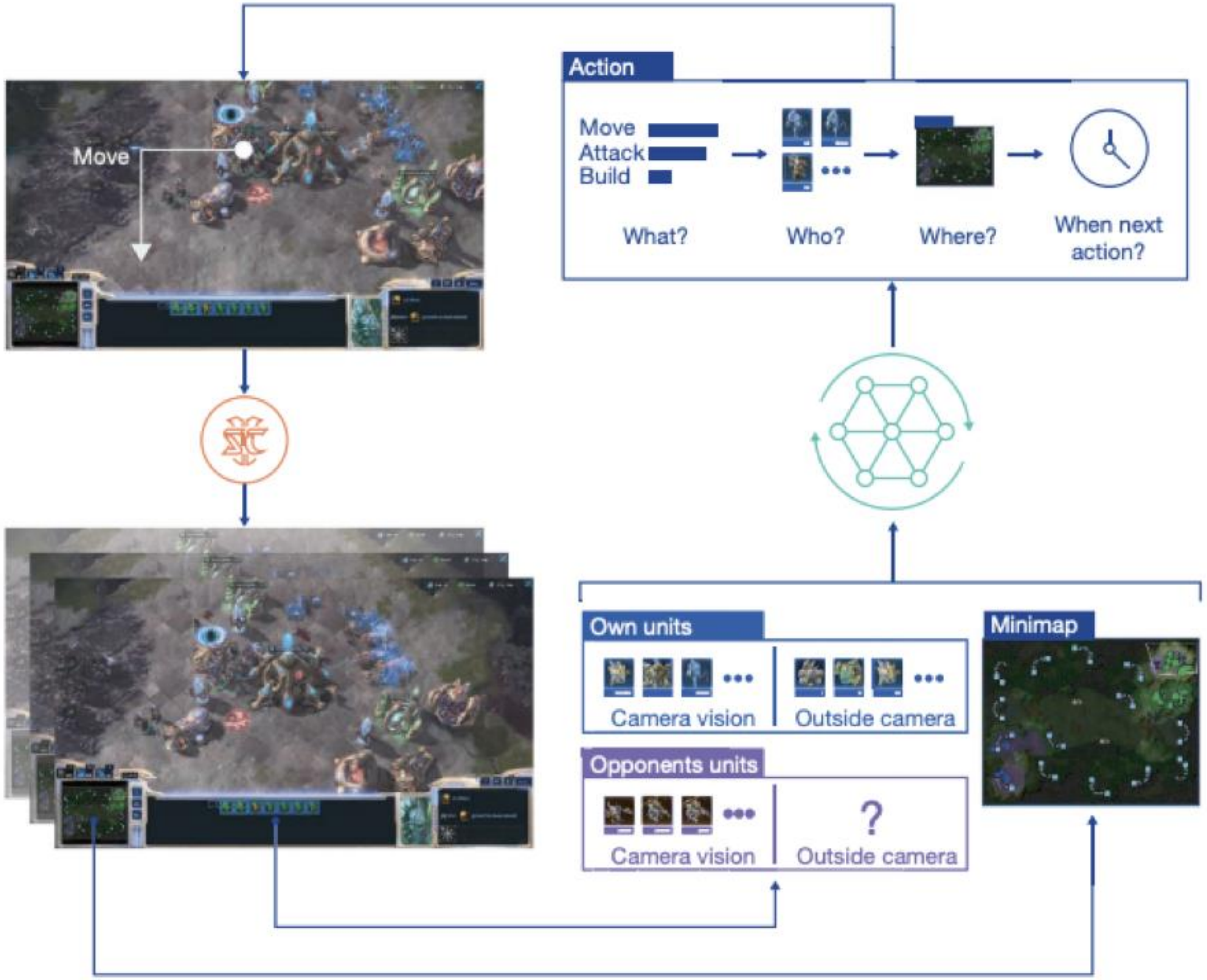
For each unit in a frame

- Unit type
- Owner
- Status
- Display type
- Position
- Number of workers
- Cool down
- Attributes
- Unit attributes
- Cargo status
- Building status
- Resource status
- Order status
- Buff status

Actions

For a unit it should have actions like

- What
  - move
  - attack
  - build
- Who
- Where
- When next action



Rewards (1/2)

Direct reward from game

- Win : +1
- Lose: -1

Pseudo-reward output along with critic network:

- the distance of agent's operation and human data statistic **z**

Rewards (2/2)

Reward is much denser in OpenAI Five at Dota2

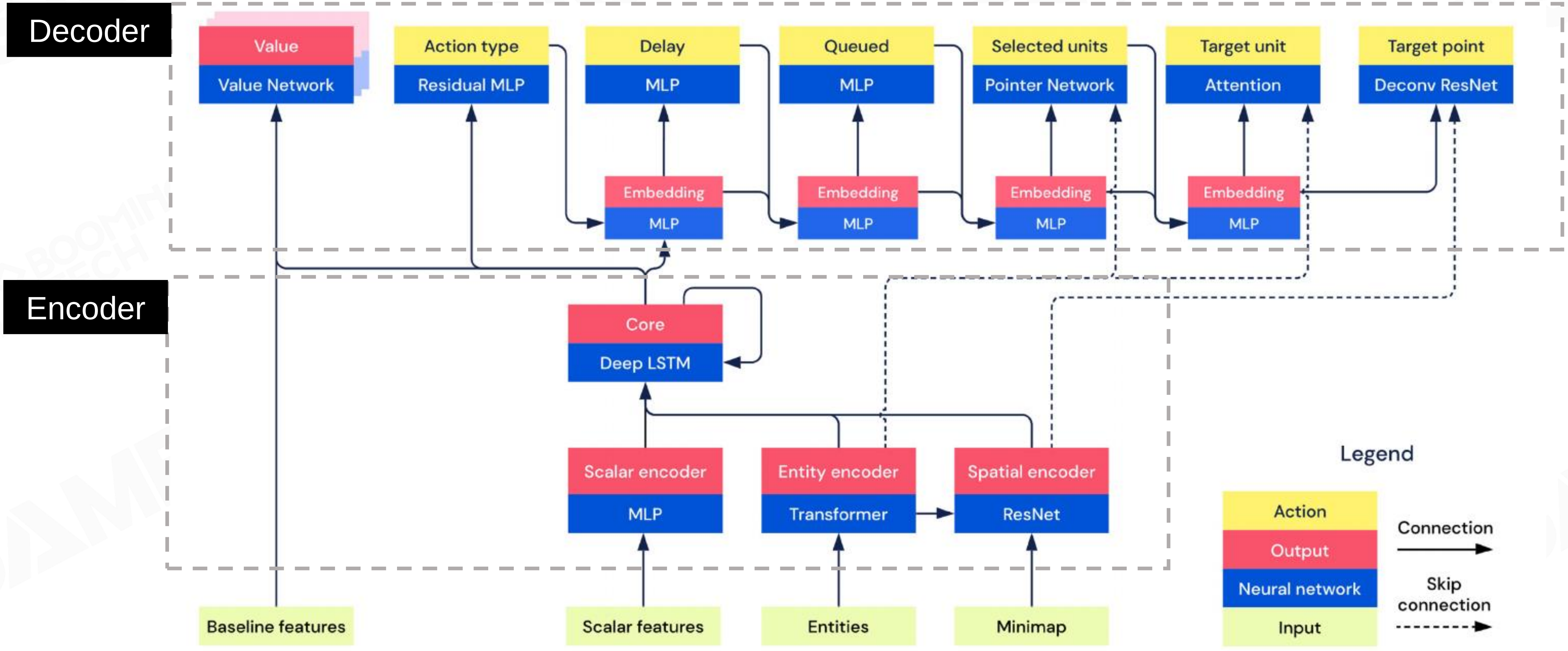
Different reward settings could help us to train different styles of agent

- Aggressive
- Conservative
- ...

Name	Reward
Win	5
Hero Death	-1
Courier Death	-2
XP Gained	0.002
Gold Gained	0.006
Gold Spent	0.0006
Health Changed	2
Mana Changed	0.75
Killed Hero	-0.6
Last Hit	-0.16
Deny	0.15
Gained Aegis	5
Ancient HP Change	5
Megas Unlocked	4
T1 Tower*	2.25
T2 Tower*	3
T3 Tower*	4.5
T4 Tower*	2.25
Shrine*	2.25
Barracks*	6
Lane Assign†	-0.15

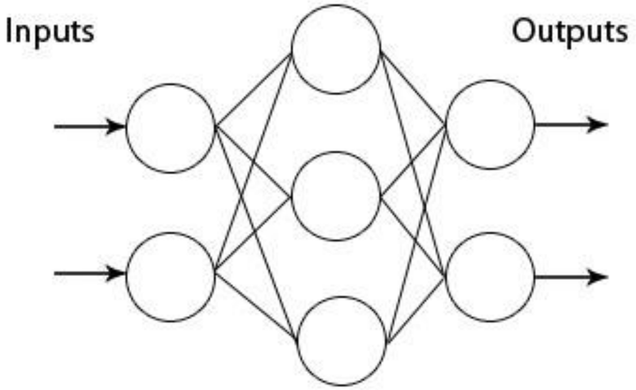
NN architectures

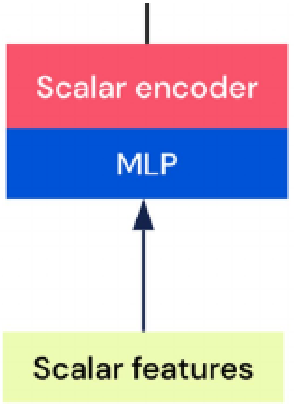
AlphaStar NN Architecture



DRL example — Multi-Layer Perceptron (MLP)

- Classical and easy to implement
- Flexible definition of the dimensions of inputs and outputs



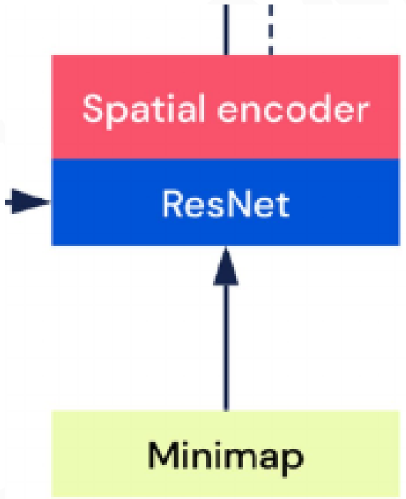


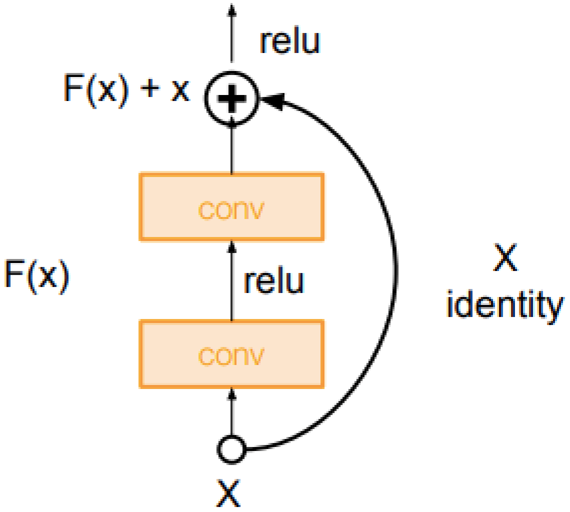
Scalar feature example

- Race
- Owned Resource
- Upgrade
- Etc.

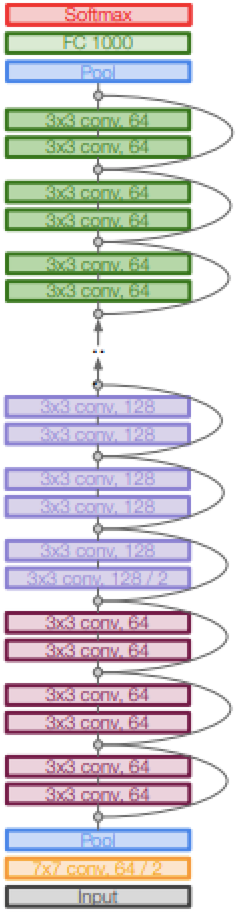
DRL example — Convolutional Neural Network (CNN)

Sensitive to image data



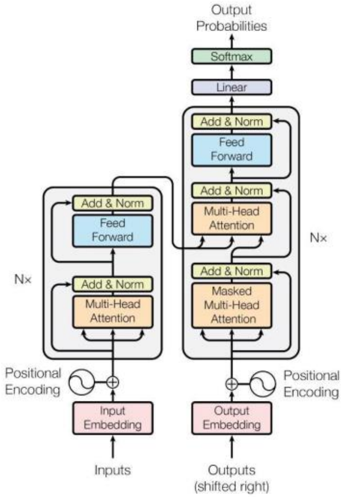


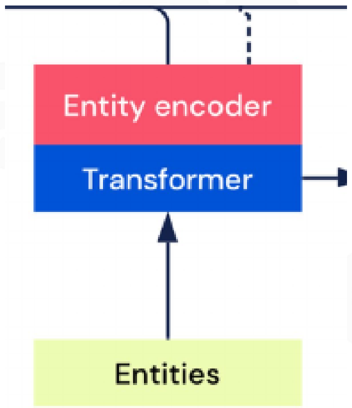
Residual block



DRL example — Transformer

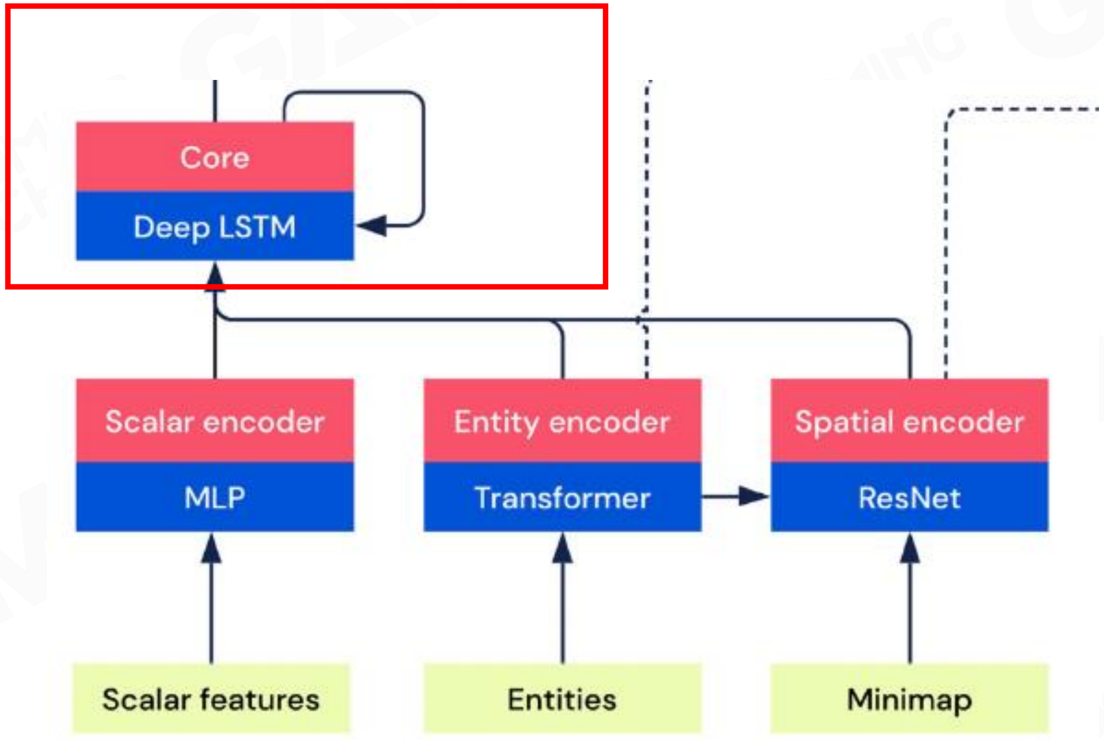
- Introduce attention mechanisms
- Uncertain length vector
- Well represent the complex feature like multi agents

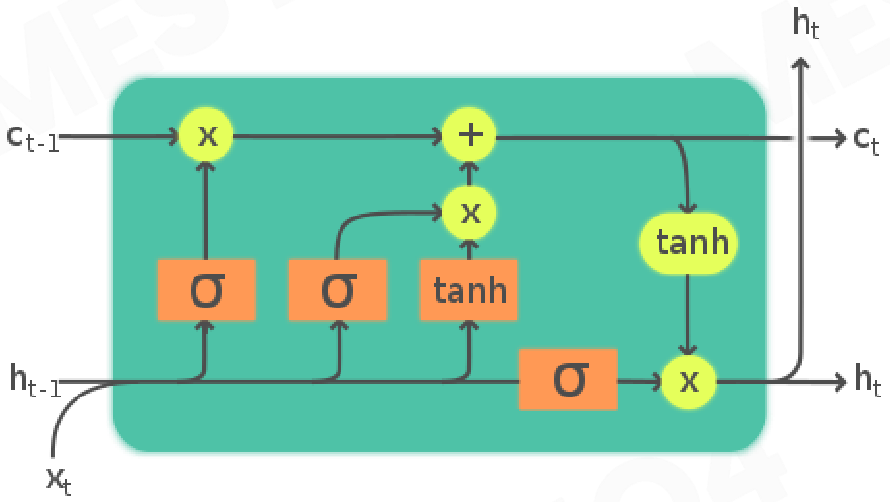




DRL example — Long-Short Term Memory (LSTM)

Enable AI to remember or forget earlier data





DRL example — NN Architecture Selection

NN Architecture selection for different type of feature

- Fixed length vector feature
  - Multi-Layer Perception
- Uncertain length vector feature
  - Long-Short Term Memory
  - Transformer
- Image feature
  - ResNet
- Raycast
- Mesh

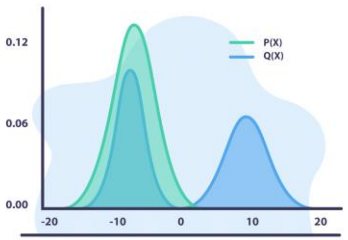
Training Strategy — Supervised learning

AlphaStar is trained via both supervised learning and reinforcement learning. It firstly learned a policy by supervised learning from human expert data

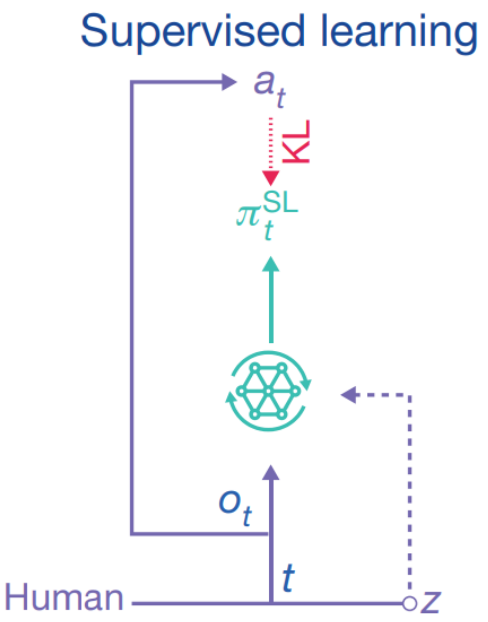
z is a statistic summary of a strategy sampled from human data (for example, a build order)

Minimize the distance (KL divergence) of agent policy and human decision distribution sampled from z

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right)$$



Kullback–Leibler Divergence



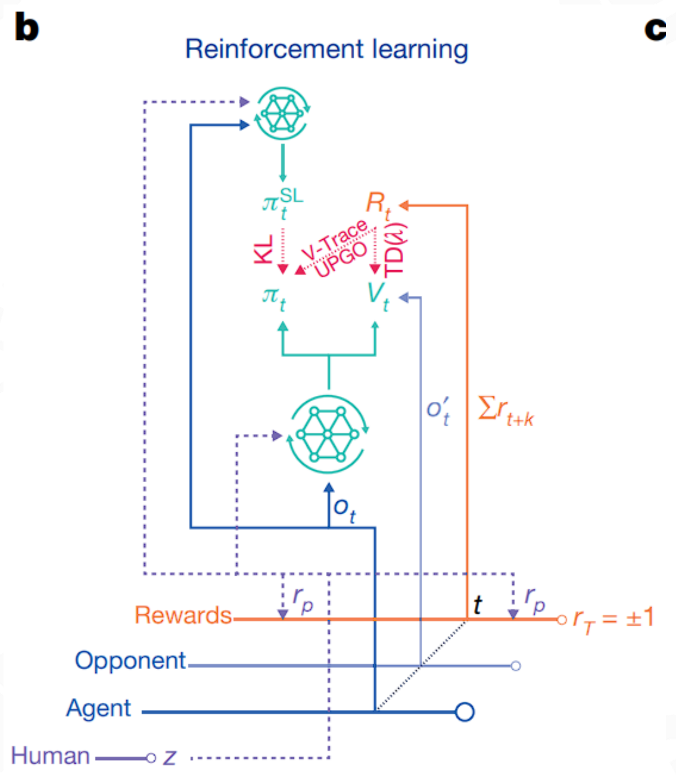
Training Strategy — Reinforcement learning

Secondly, it took RL technique to improve the SL policy

TD(λ), V-trace, UPGO are specific Reinforcement learning methods to improve actor network and critic network.

The KL degree towards old SL policy would also be considered

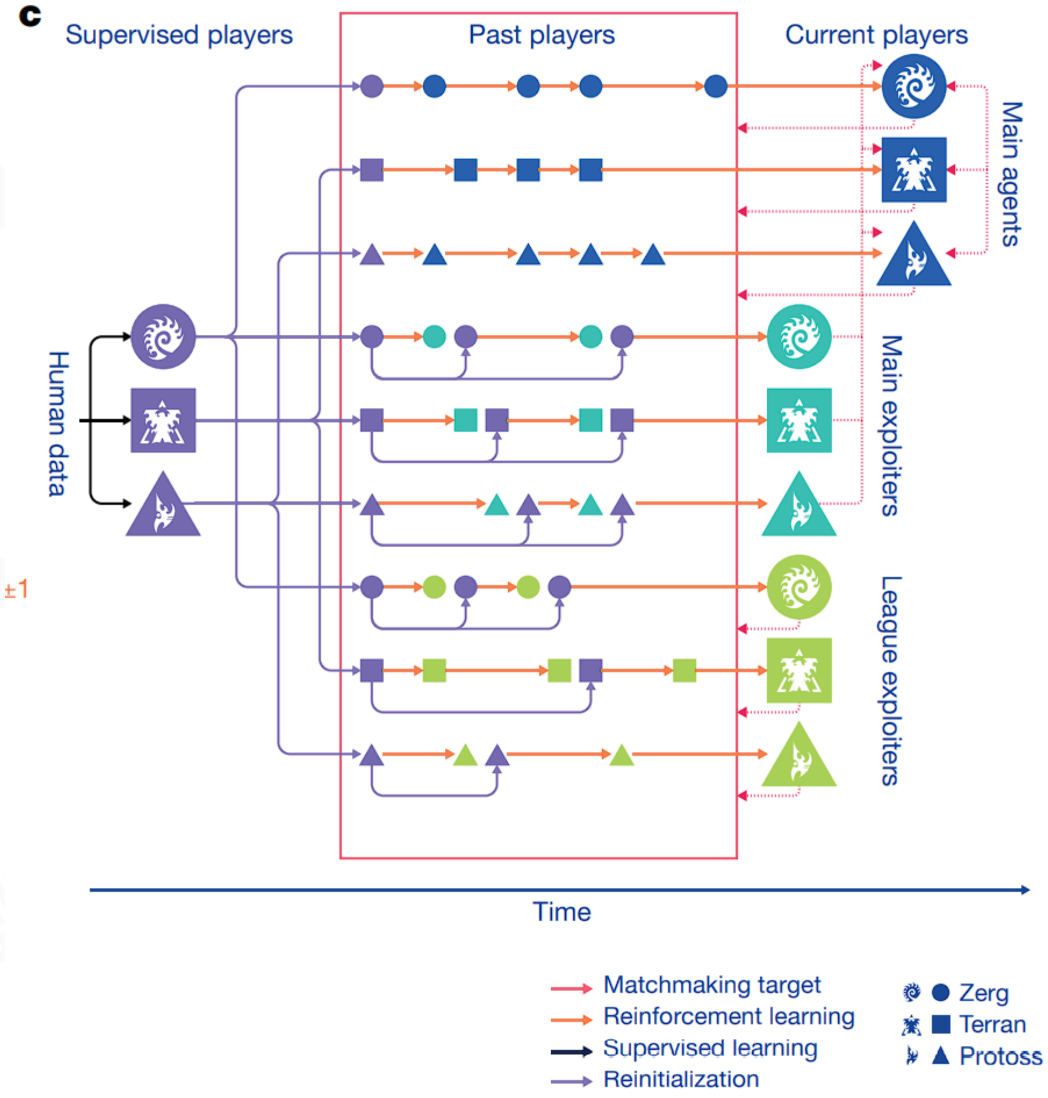
These tricks improved the policy and made it more human-like



Train the Agent — Self Play & Adversarial

In AlphaStar three pools of agents attend training initialized from SL policy

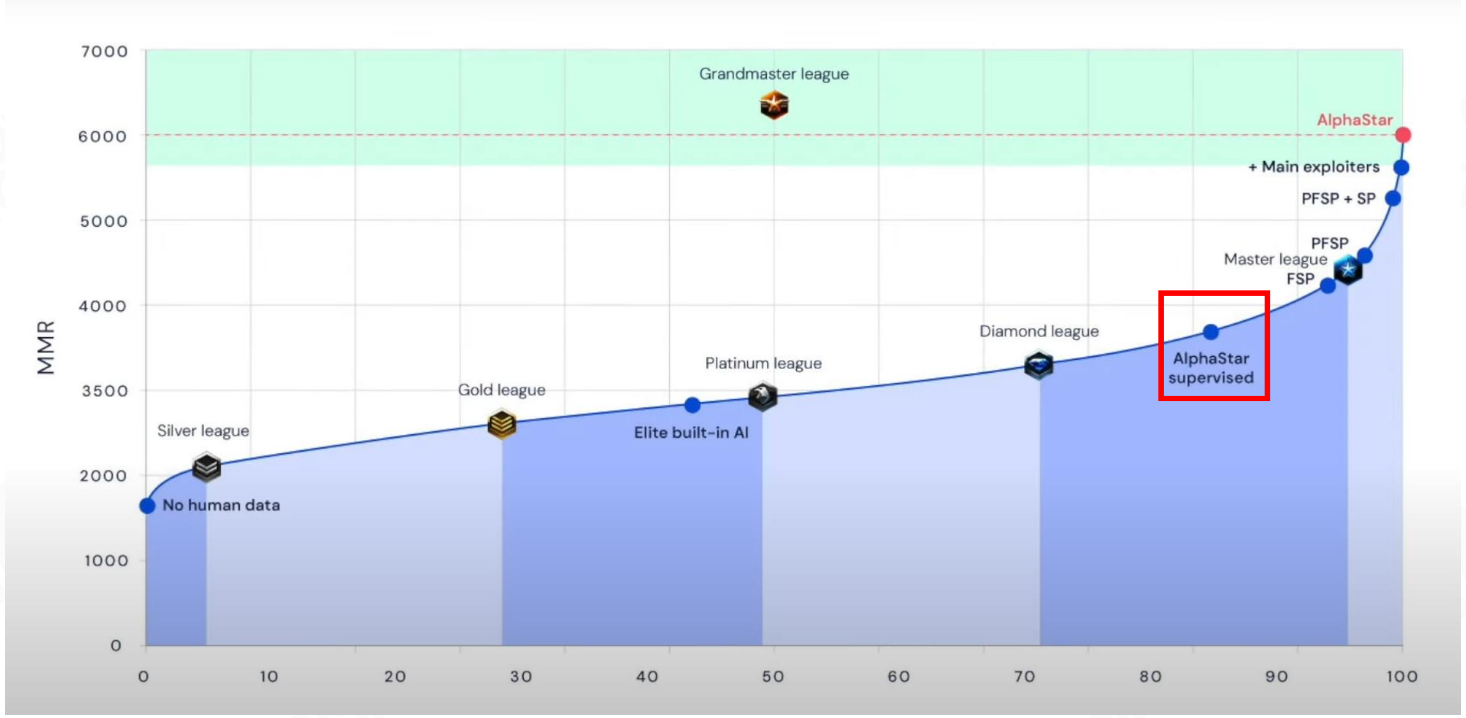
- Main agents [MA]
  - Goal: most robust and output
  - Self-play (35%)
  - Against past LE and ME agents(50%)
  - Against past MA agents(15%)
- League exploiters[LE]
  - Goal: find weakness of past all agents (MA, LE, ME)
  - Against all past agents (MA, LE, ME)
- Main exploiters [ME]
  - Goal: find weakness of current MA agent
  - Against current MA agent



RL or SL? —— SL analysis

Supervised Learning needs high quality data, and sometimes behaves well too

- It behaves like human
- But may not outperform human expert data
- Human data is unbalanced
- Sometimes there is not enough data



RL or SL? —— RL analysis

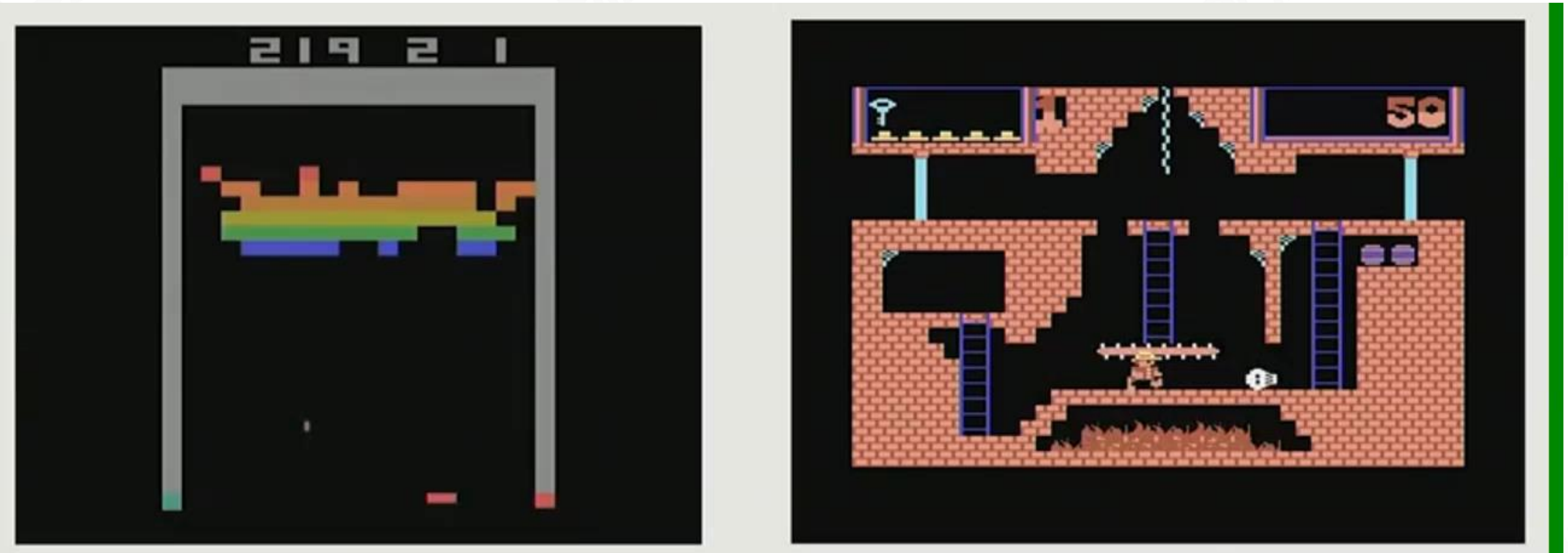
Reinforcement Learning is usually considered as the **optimal** solution, **however**

- Training a RL model is tough
  - The model is hard to converge
  - The game environment for training is also a huge development project
  - The data collection process could be slow
  - And the behavior maybe unnatural

RL or SL? —— Dense reward

What makes a good problem for RL

Dense reward | Sparse reward



Breakout | Montezuma’s Revenge

RL or SL? —— Summary

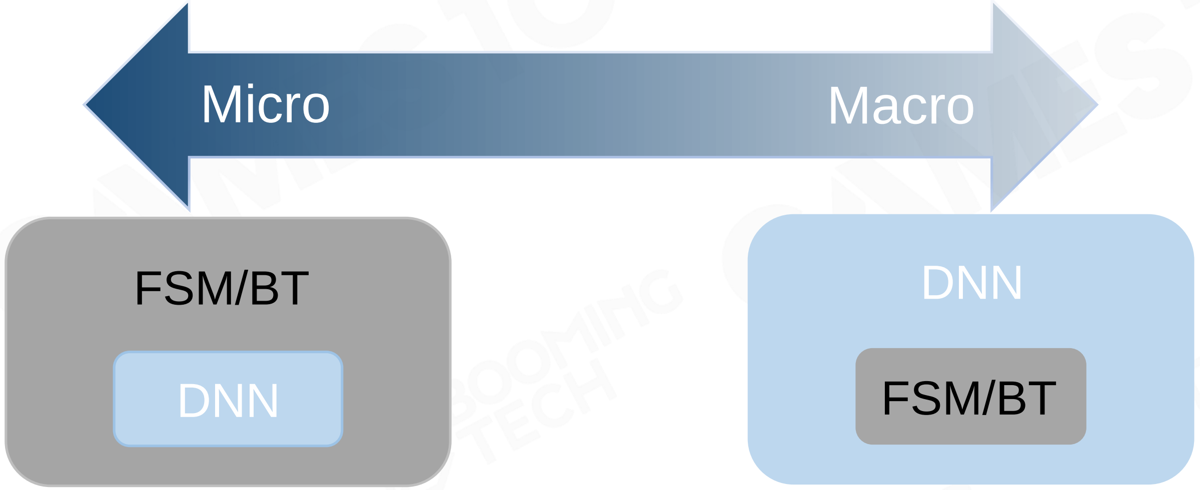
Situation for SL	Situation for RL
• Easy to get data	• Needs to outperform the master level
• Needs to perform like human	• Enough budget
	• Data is unavailable
	• Dense reward

Hybrid

Machine Learning is powerful.

But it cost much too. For example, DeepMind spends **250 million** dollars to finish alpha star and a replication needs **13 million dollars**

We often need to make a tradeoff that **place DNN on the human-like points**(a part of the whole combat).



Navigation for ships

Evaluation for a combat

Ages of Empire IV

References

HTN

- Humphreys T. Exploring HTN planners through example, Game AI Pro 360. CRC Press, 2019: 103-122. https://www.gameaipro.com/GameAIPro/GameAIPro_Chapter12_Exploring_HTN_Planners_through_Example.pdf
- An overview of hierarchical task network planning. Georgievski I, et al. arXiv preprint arXiv:1403.7426, 2014. https://arxiv.org/abs/1403.7426
- Advanced Real-Time Hierarchical Task Networks: A New Approach. Tomohiro M, et al. Square enix. GDC 2021. https://gdcvault.com/play/1027232/AI-Summit-Advanced-Real-Time

GOAP

- Enhanced NPC behaviour using goal oriented action planning. Long E. Master's Thesis, School of Computing and Advanced Technologies, University of Abertay Dundee, Dundee, UK, 2007. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.131.8964&rep=rep1&type=pdf
- Goal-oriented action planning: Ten years old and no fear! Chris C, et al. Crystal Dynamics. GDC 2015. https://gdcvault.com/play/1022019/Goal-Oriented-Action-Planning-Ten
- AI Action Planning on Assassin's Creed Odyssey and Immortals Fenyx Rising. Simon G. Ubisoft. GDC 2021. https://gdcvault.com/play/1027004/AI-Action-Planning-on-Assassin

MCTS

- A survey of monte carlo tree search methods. Browne C B, et al. IEEE Transactions on Computational Intelligence and AI in games, 2012, 4(1): 1-43. https://www.researchgate.net/publication/235985858_A_Survey_of_Monte_Carlo_Tree_Search_Methods
- Enhancements for real-time Monte-Carlo tree search in general video game playing. Soemers D, et al. 2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, 2016: 1-8. https://ieeexplore.ieee.org/abstract/document/7860448/
- Action guidance with MCTS for deep reinforcement learning. Kartal B, et al. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. 2019, 15(1): 153-159. https://ojs.aaai.org/index.php/AIIDE/article/view/5238

Machine Learning (1/2)

- Great Chinese tutorial on reinforcement learning: Easy-RL, 人民邮电出版社, 2022, https://github.com/datawhalechina/easy-rl
- Classic English textbook on reinforcement learning: Reinforcement Learning: an introduction, MIT press, 2018, https://web.Stanford.edu/class/psych209/Readings/SuttonBartoIPRLBook2ndEd.pdf
- Classic course on reinforcement learning: Stanford Course CS234: https://web.Stanford.edu/class/cs234/
- Key papers on reinforcement learning selected by OpenAI: https://spinningup.openai.com/en/latest/spinningup/keypapers.html

Machine Learning (2/2)

- Classic handbook on deep learning: https://github.com/janishar/mit-deep-learning-book-pdf
- An introduction to convolutional neural networks, O'Shea K, et al. arXiv preprint arXiv:1511.08458, 2015. https://arxiv.org/pdf/1511.08458.pdf
- Understanding LSTM – a tutorial into long short-term memory recurrent neural networks, Staudemeyer R C, et al. arXiv preprint arXiv:1909.09586, 2019: https://arxiv.org/abs/1909.09586
- Attention is all you need. Vaswani A, et al. Advances in neural information processing systems, 2017, 30. : https://arxiv.org/abs/1706.03762

Machine Learning Game Applications (1/2)

- Atari: Mnih V, et al. Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602, 2013. https://arxiv.org/pdf/1312.5602.pdf
- Honor of Kings : Ye D, et al. Mastering complex control in moba games with deep reinforcement learning. Proceedings of the AAAI Conference on Artificial Intelligence. 2020, 34(04): 6672-6679. https://ojs.aaai.org/index.php/AAAI/article/view/6144
- Dota2: Berner C, et al. Dota 2 with large scale deep reinforcement learning. arXiv preprint arXiv:1912.06680, 2019. https://arxiv.org/pdf/1912.06680.pdf
- Starcraft2: Vinyals O, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning[J]. Nature, 2019, 575(7782): 350-354. https://www.nature.com/articles/s41586-019-1724-z.pdf

Machine Learning Game Applications (2/2)

- DRL for navigation: Deep Reinforcement Learning For Navigation. Maxim P, et al. Ubisoft. GDC 2021. https://gdcvault.com/play/1027382/Machine-Learning-Summit-Deep-Reinforcement
- Machine Learning in game engine: It's Complicated: Getting ML inside a AAA Engine. Adrien L, et al. Ubisoft. GDC 2022. https://gdcvault.com/play/1027851/Machine-Learning-Summit-It-s
- Age of Empires IV: 'Age of Empires IV': Machine Learning Trials and Tribulations. Guy L, et al. Microsoft. GDC 2022. https://gdcvault.com/play/1027936/AI-Summit-Age-of-Empires
- Imitation Learning case: Buffing Bots with Imitation Learning. Niels J, et al. modl.ai. GDC 2022. https://gdcvault.com/play/1027942/AI-Summit-Buffing-Bots-with

Lecture 17 Contributors

- 一将
- yunhe
- 大喷
- 普普
- Olorin
- 灰灰
- 喵小君
- 蓑笠翁

Q&A

Enjoy ;) Coding

Course Wechat

Follow us for further information