29th Soar Workshop

Learning to play Mario

Shiwali Mohan
University of Michigan, Ann Arbor

Outline

- Why games?
- Domain
- Approaches and results
- Issues with current design
- What's next…

Why computer games? Why Mario?

- Computer games
  - Complex, can have really large, continuous state and action spaces.
  - Reasoning and learning required at many levels
    - Sensory-motor primitives
    - Path planning
    - Strategy
  - Knowledge learned in one level should be applicable in other levels of the game.
  - Exciting to watch Soar play!

- Infinite Mario
  - Parameterized domain from RL Competition 2009
  - Easy interface (RL-glue)

Domain

- Visual space
  - 16 x 22 tiles in visual scene, each tile can have 13 different values
  - 9 kinds of monsters, can be anywhere, agent knows their exact location, speed, type
- Reasonable action space
  - Left, Right, Stay, Jump and Speed Toggle (12 combinations).
- Reward
  - 100 to reach finish line
  - -1 for every step in the environment
  - +1 killing monster, taking coin etc
  - -10 for dying

- Not real time.
- Sample Agent
  - Memory of last episode
- Levels
  - 0-3

How should states be represented?

Simple RL - agent

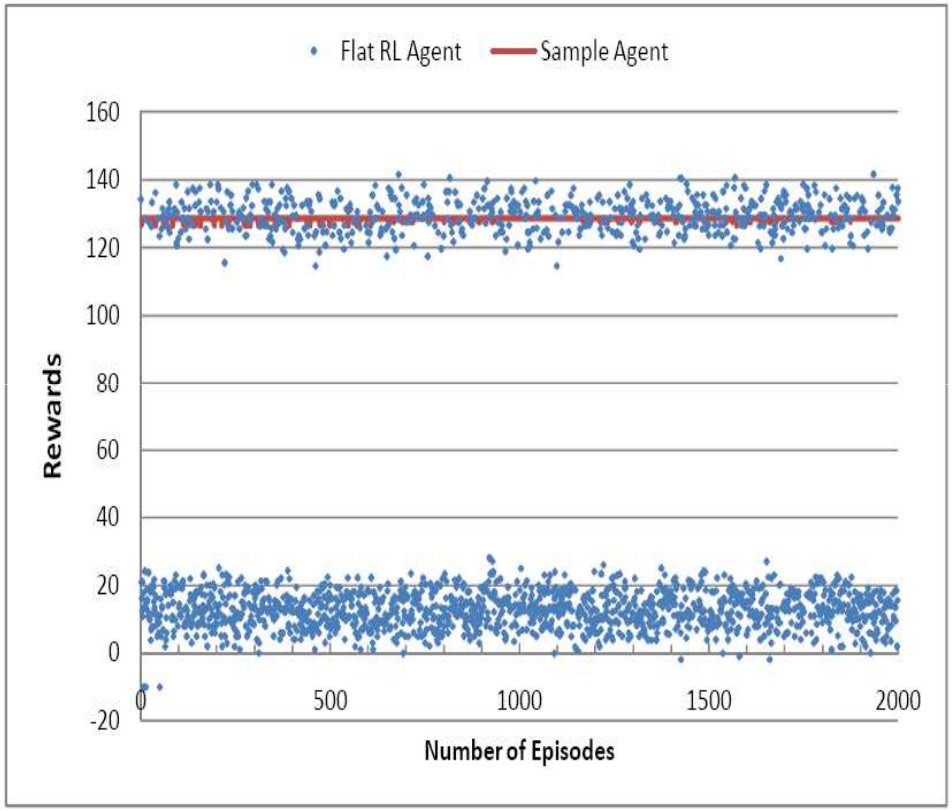
State – [5 x 5] Tiles around Mario

Action space – Primitive Actions,- left, right, still jump, speed

Simple RL agent - Results

- Reasons
  - Huge state space!
  - Rewards too far into future.
  - A large number of steps required to complete on episode

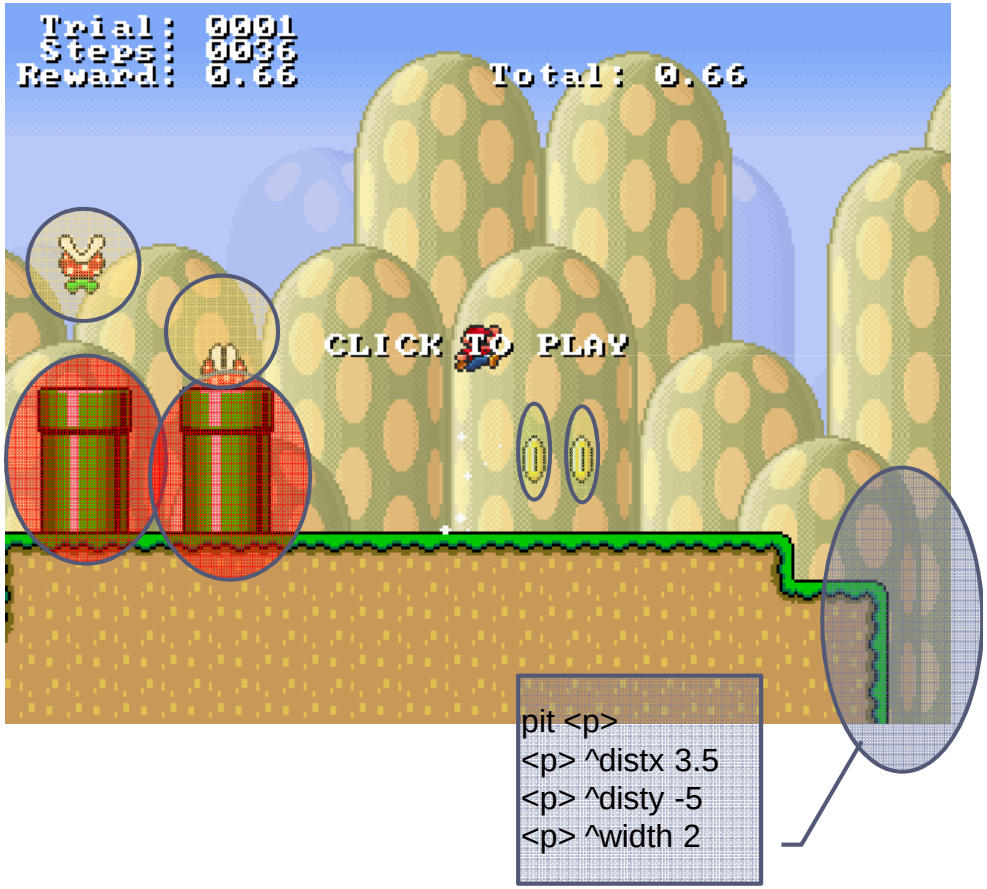
Also, a policy if ever learned, might not help in other instances of the game.



Different way to look at the problem…

- Object oriented representation* of state
  - From low level tile by tile representation to a view composed of objects and their relationships with the agent.
  - From "*the tile at position (x,y) is of type t*" or "*the agent is at tile (x,y)*" to "*pit P is at a distance of 3.5 tiles on x -axis*".
  - Features like pits, pipes, platforms etc extracted from the lower level data using simple heuristics.

    Can be accounted for as background knowledge.



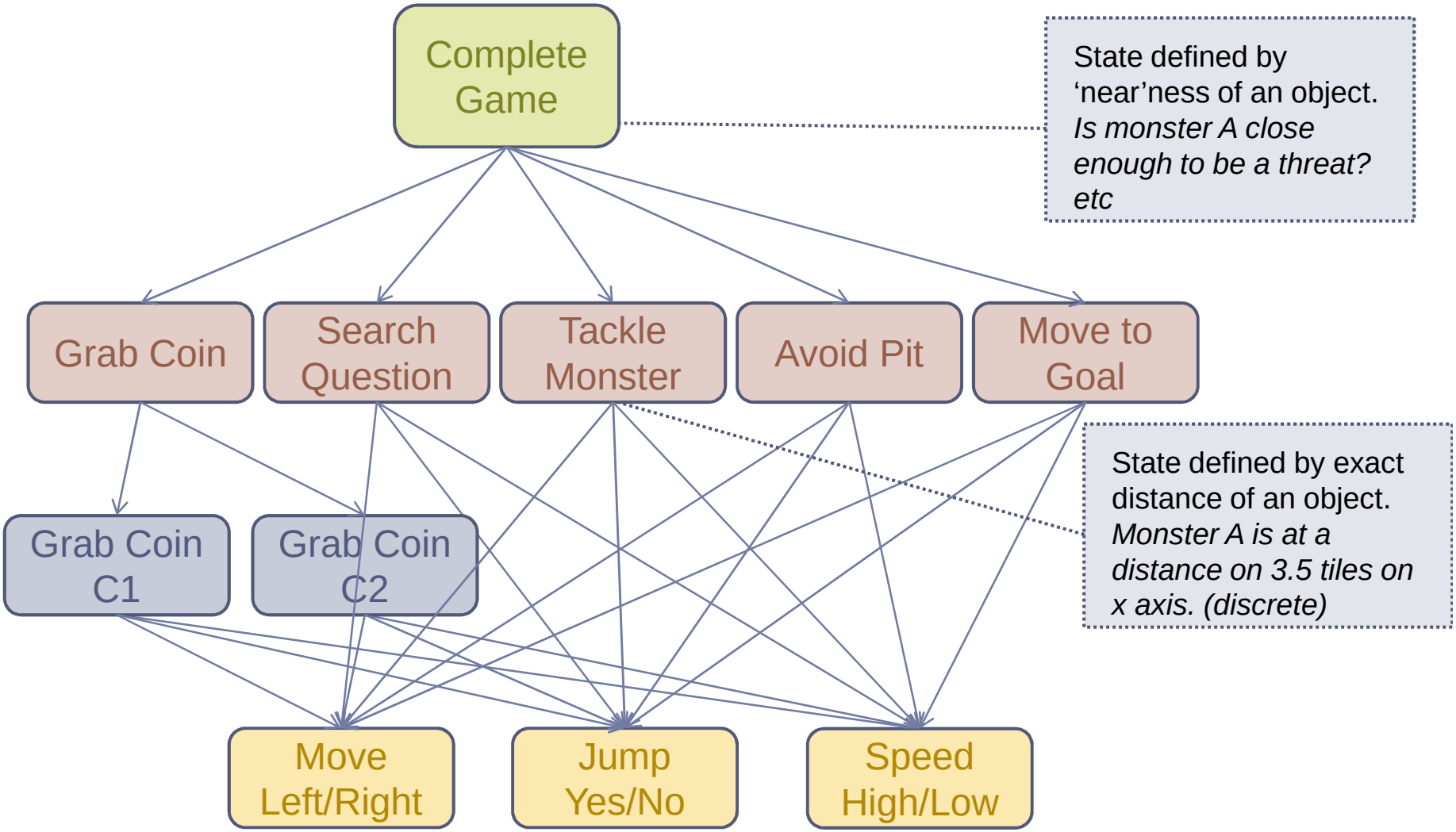
*Diuk, Cohen and Littman, "An object-oriented representation for efficient reinforcement learning", in proceedings of ICML 2008

Operator/Action abstraction

- Make a distinction between key-stroke level operators (KLOs) and functional level operators (FLOs)
  - KLOs – primitive actions (left, right jump etc in present case)
  - FLOs – collection of KLOs executed in a sequence to perform a specific task
- Key observations from GOMS* analysis on HI-Soar
  - Games have a limited number for FLOs (depends different categories of objects present)
  - Human experts decide between different FLOs using very local and immediate conditions / features (like distance from monsters, coins etc)
    - Example FLOs – tackle_monster, grab_coin, search_question
- How does this help?
  - Provides structure to the problem.
  - When agent executes a FLOs, it has an immediate goal to attend like getting rid of the monster.
    - Learning is easier, similar to HRL

*John B. E. Extensions of GOMS analyses to expert performance requiring perception of dynamic visual and auditory information. In proceedings of CHI 1990

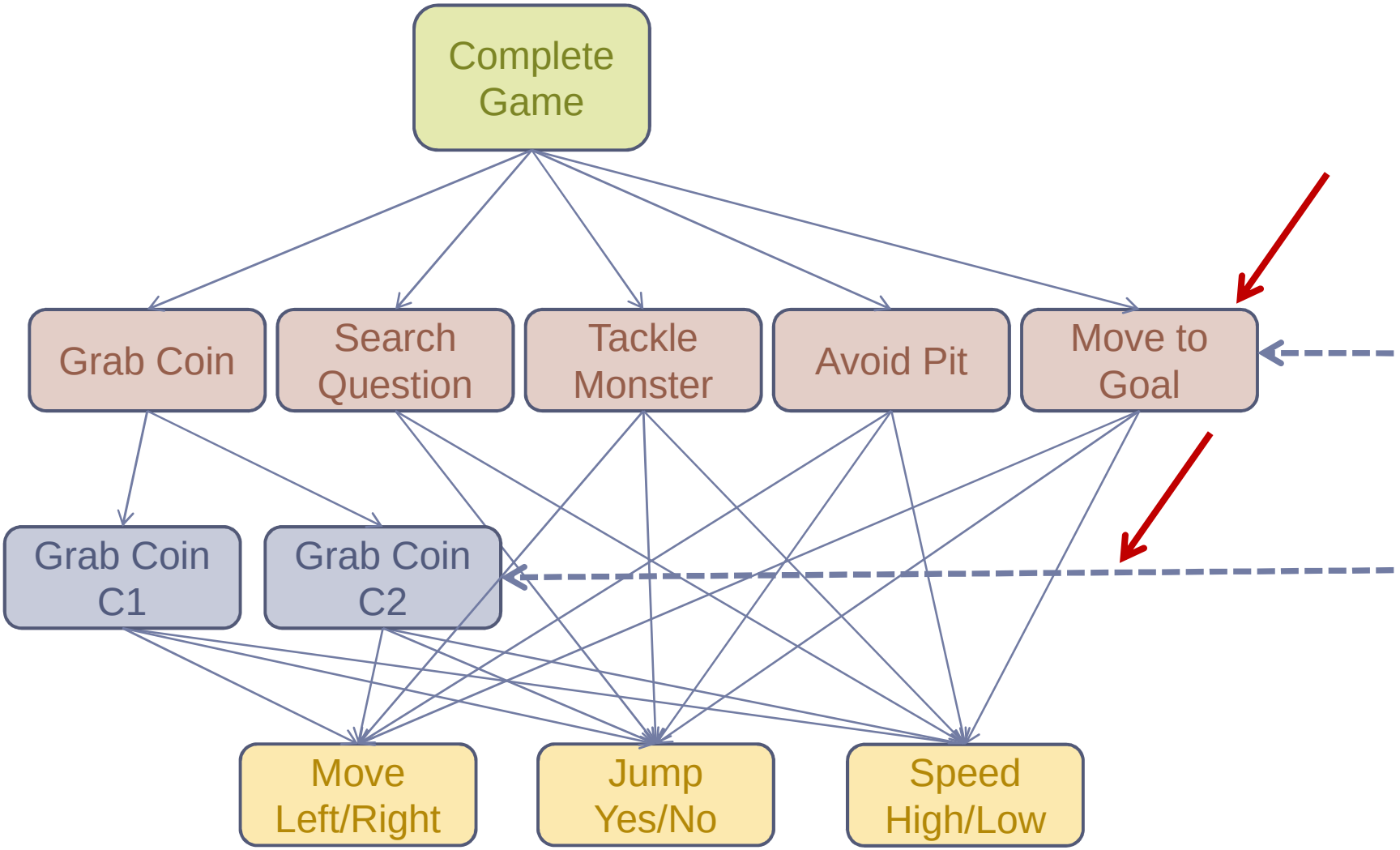
Operator Hierarchy

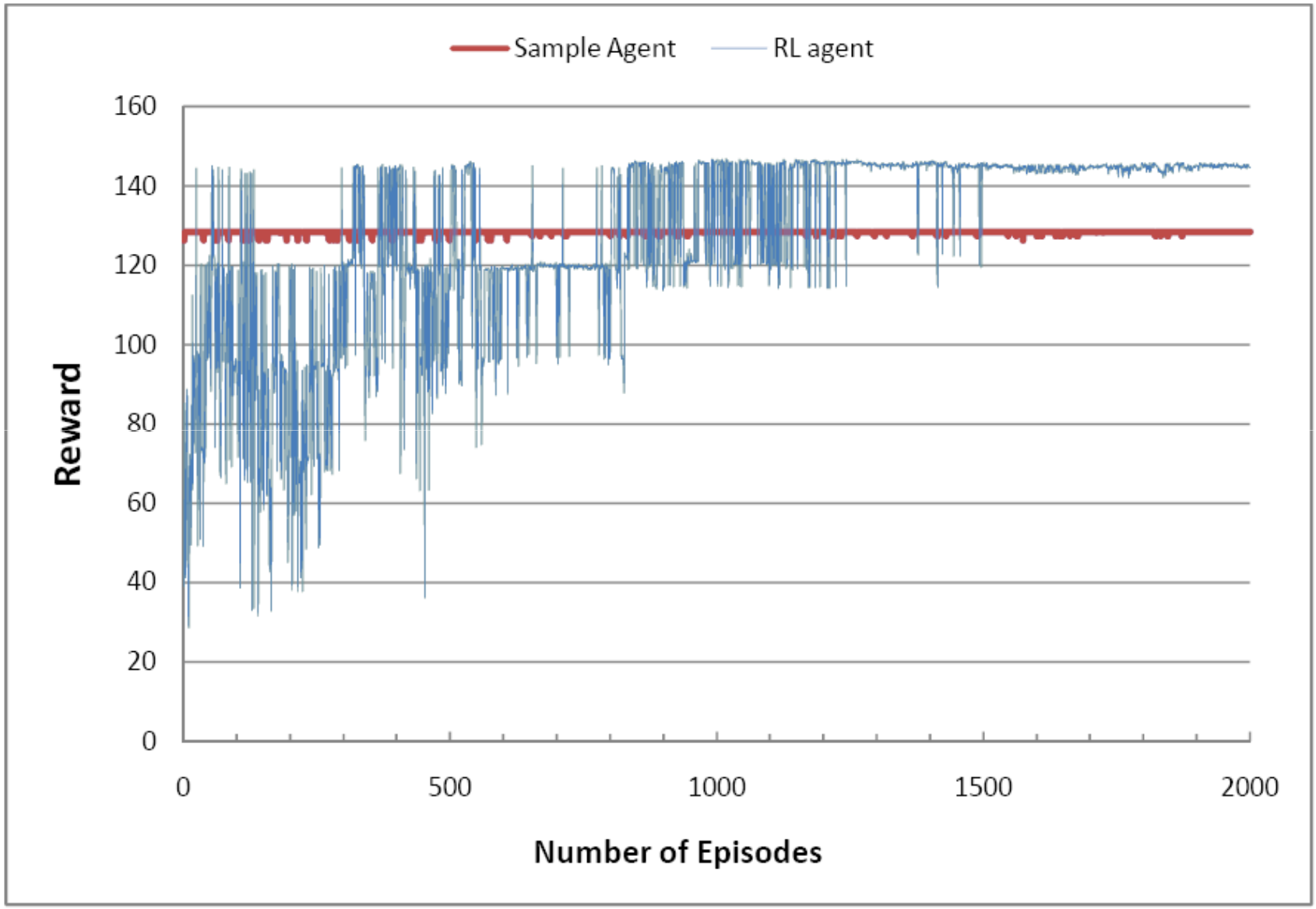


How it works…

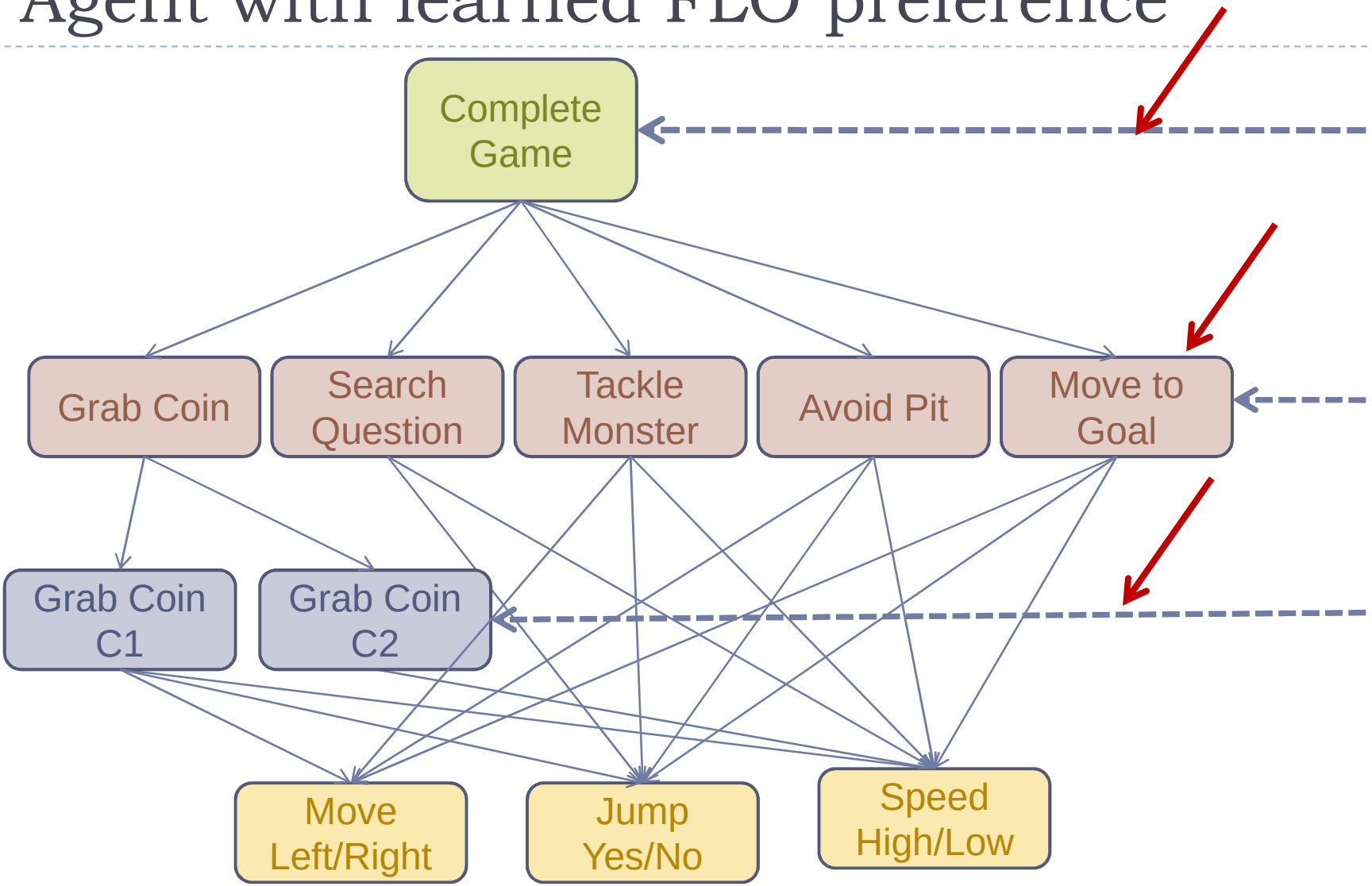
- Objects from the sensory input are extracted and elaborated to have a relational (with Mario) representation.
- All objects in “close” vicinity cause FLO proposals.
- The agent makes a selection between these operators based on hard coded/ learned preferences.
- Agent goes into a sub-goal to execute the FLO; FLO proposes lower, keystroke-level operators.
- Primitive actions are executed, environment steps, produces a reward and next set of sensory data.

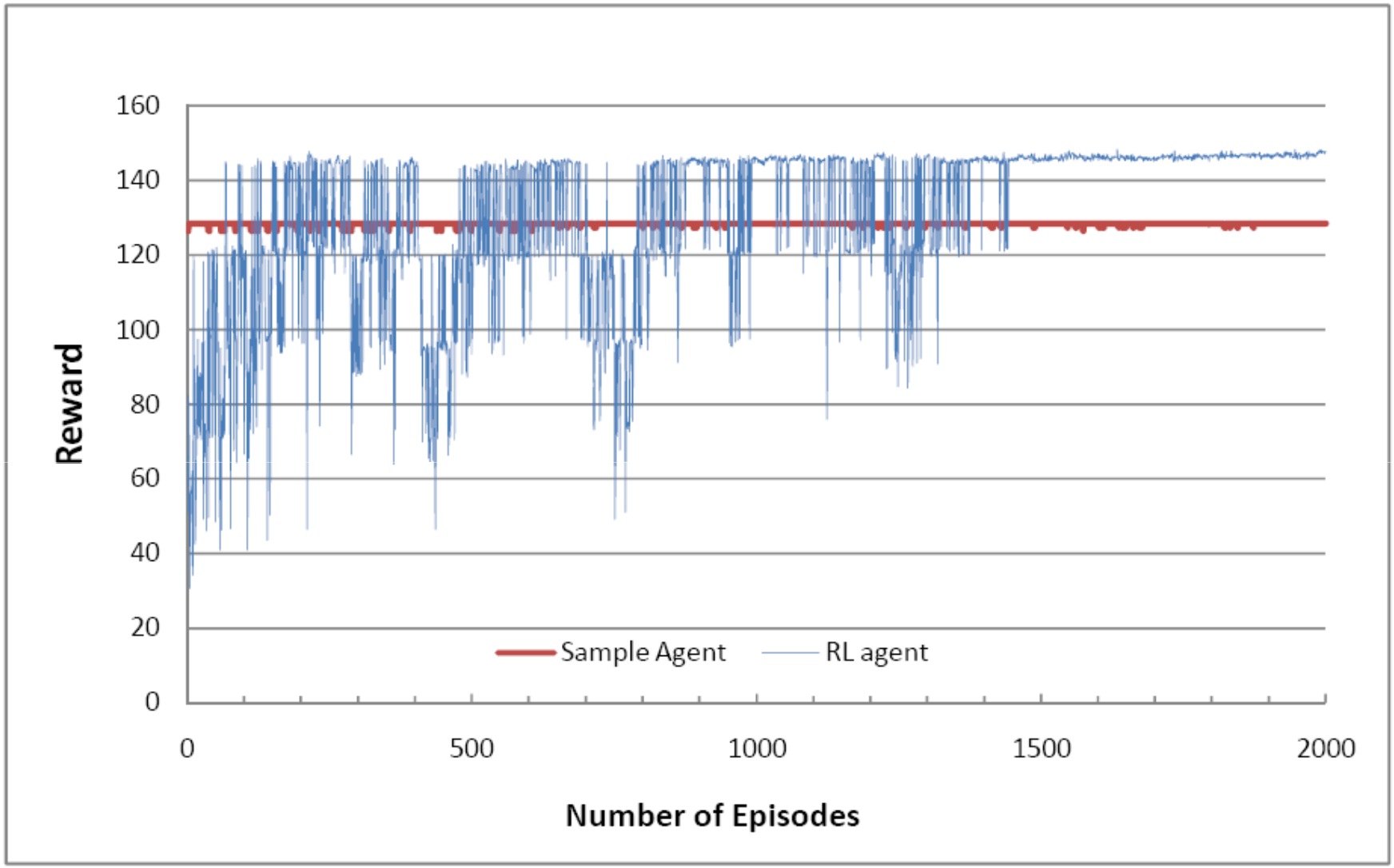
Agent with hard-coded FLO preferences

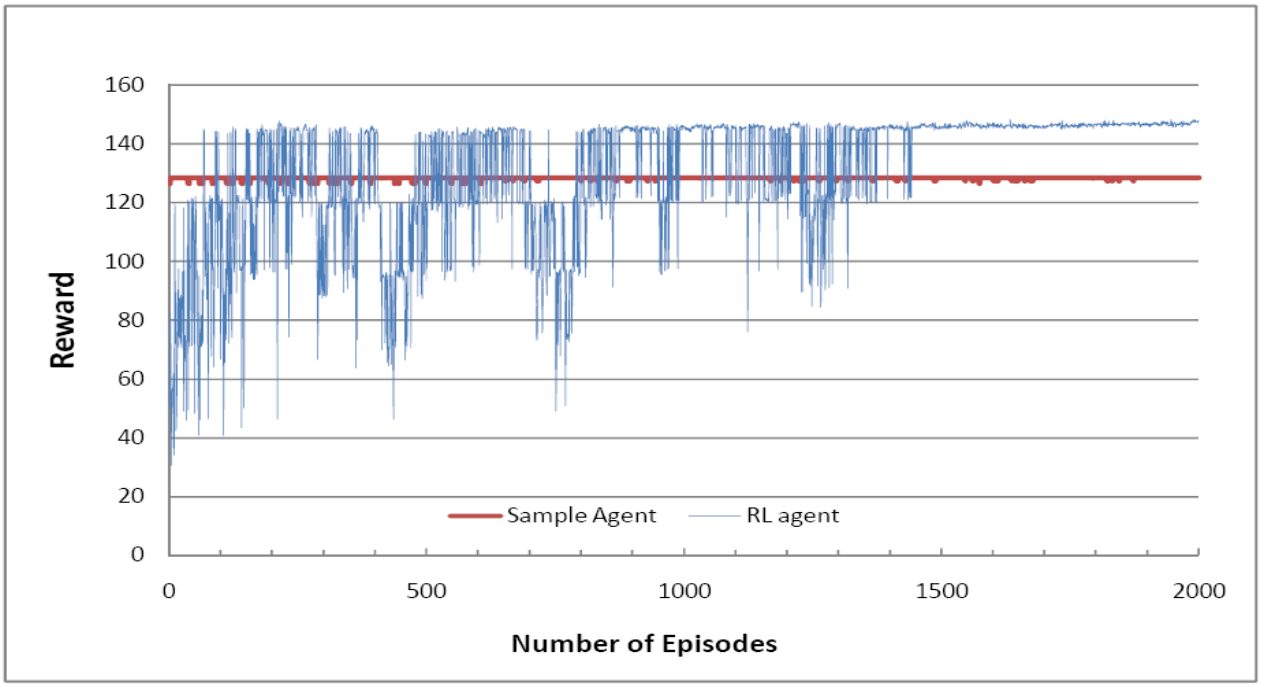



Sample Agent
RL agent
160
140
120
100
80
60
40
20
0
Reward
0
500
1000
1500
2000
Number of Episodes

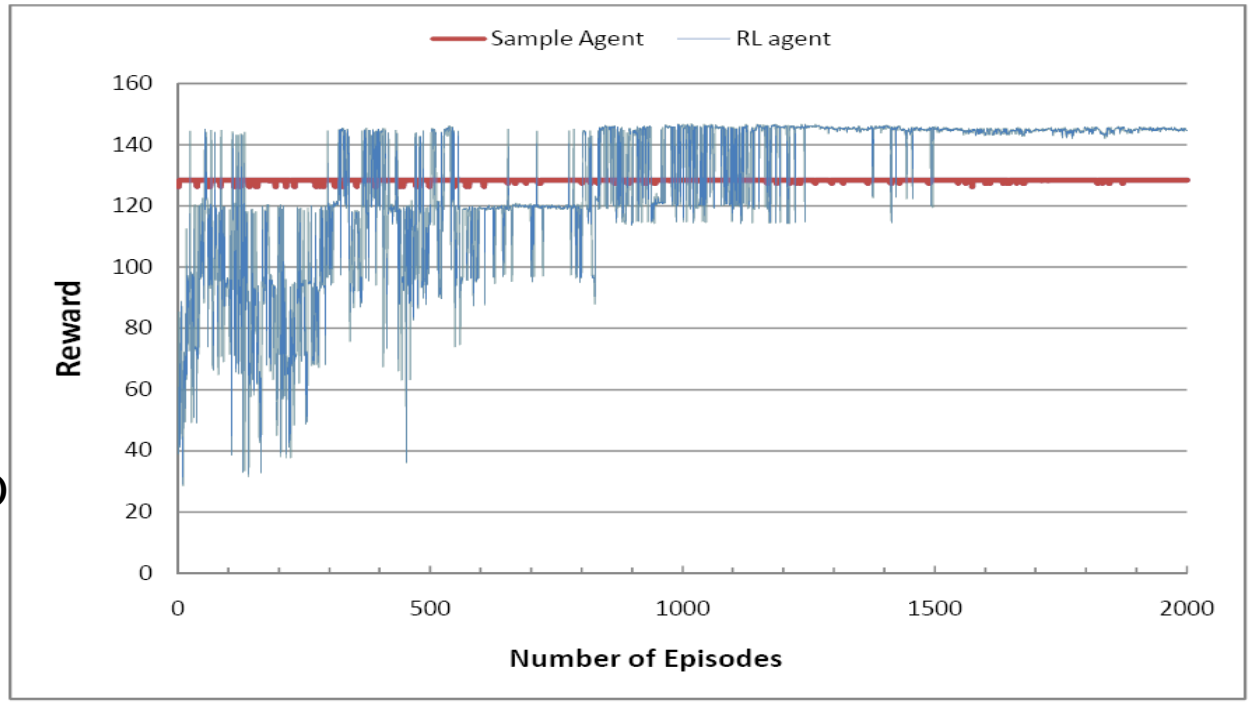
Agent with learned FLO preference




160
140
120
100
80
60
40
20
0
Reward
Sample Agent
RL agent
0
500
1000
1500
2000
Number of Episodes



Hard coded FLO preferences.

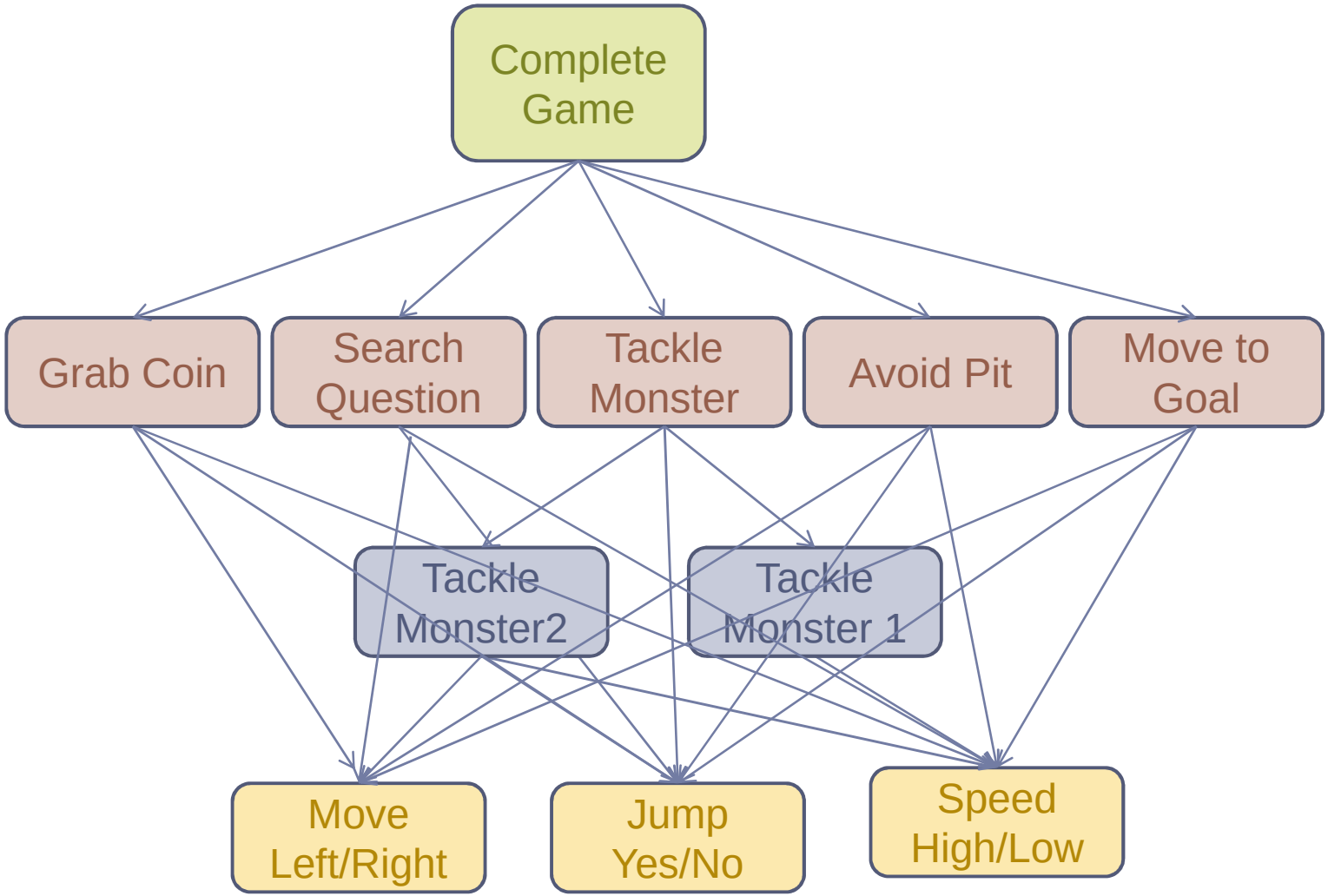


Learned FLO preferences.

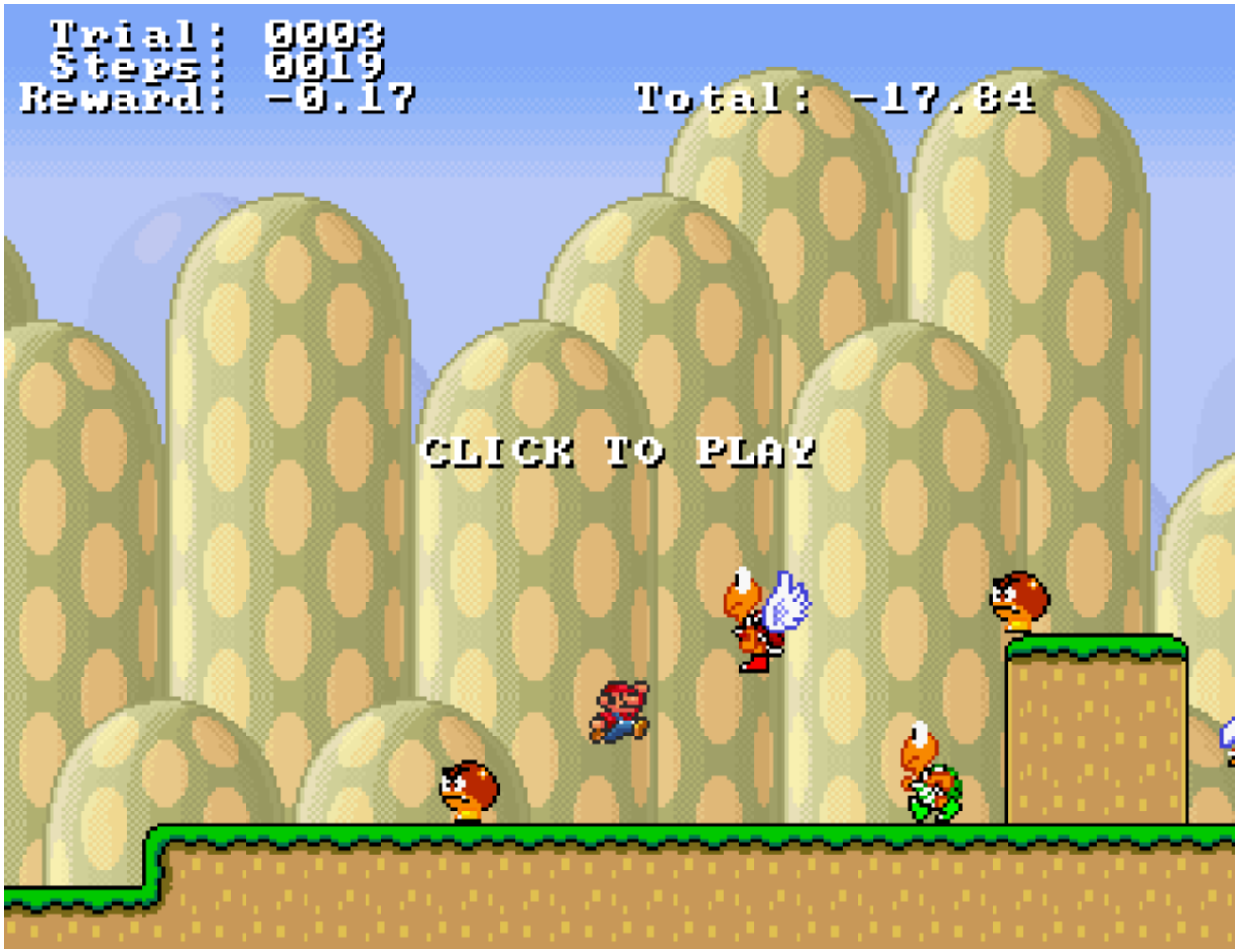
How good is the current design?

- Good at simple scenarios where decisions affected by only one object in the vicinity.
  - In case there are more than one objects, the preference is clear,
    - Dealing with monster has higher preference than collecting coins.
- In case of multiple objects have to be considered together, the agent fails to learn a good policy

Trouble


Complete Game
Grab Coin
Search Question
Tackle Monster
Avoid Pit
Move to Goal
Tackle Monster2
Tackle Monster 1
Move Left/Right
Jump Yes/No
Speed High/Low

Example scenarios



Next question to ask…

- When operators (KLOs) have conflicting preferences due to different objects, what constitutes a good policy? How can it be learned?
- What is a ‘conflict’? How can it be detected?

Some ideas -

- 'Learn' more specific rules.
  - For Objects A and B, the agent has learned independent policies for A, B.
  - If both A and B are encountered, previously leaned policies are split as policies for -
    - A ^ !B
    - !A ^ B
    - A ^ B (to be learned from more experience)

- Move beyond Reinforcement Learning –
  - The situation causes an impasse.
  - Agent uses other tools like episodic memory or mental imagery and spatial reasoning to deal with the current situation and chunks the information.

Finally…

- Nuggets
  - Relational, object oriented representations used in reinforcement learning.
  - Imposing hierarchy facilitates better learning, better organization.
  - Generalized learning

- Coal
  - Multiple object affecting decision still a problem.
  - Using templates makes it slow (can not be used in a real time games)
  - Using gp causes it to produce large (~million) rules quite a few of which are never used.
  - Continuous values!