

Adaptive reinforcement learning agents in RTS games



ERIC KOK – UTRECHT UNIVERSITY
EMKOK@STUDENTS.CS.UU.NL
WWW.EKOK.NL/TECH/BOS22APL/

SUPERVISED BY
DR. FRANK DIGNUM AND JOOST WESTRA MSC

Introduction



- Challenges in playing RTS games
- Bos Wars as platform for game research



Introduction

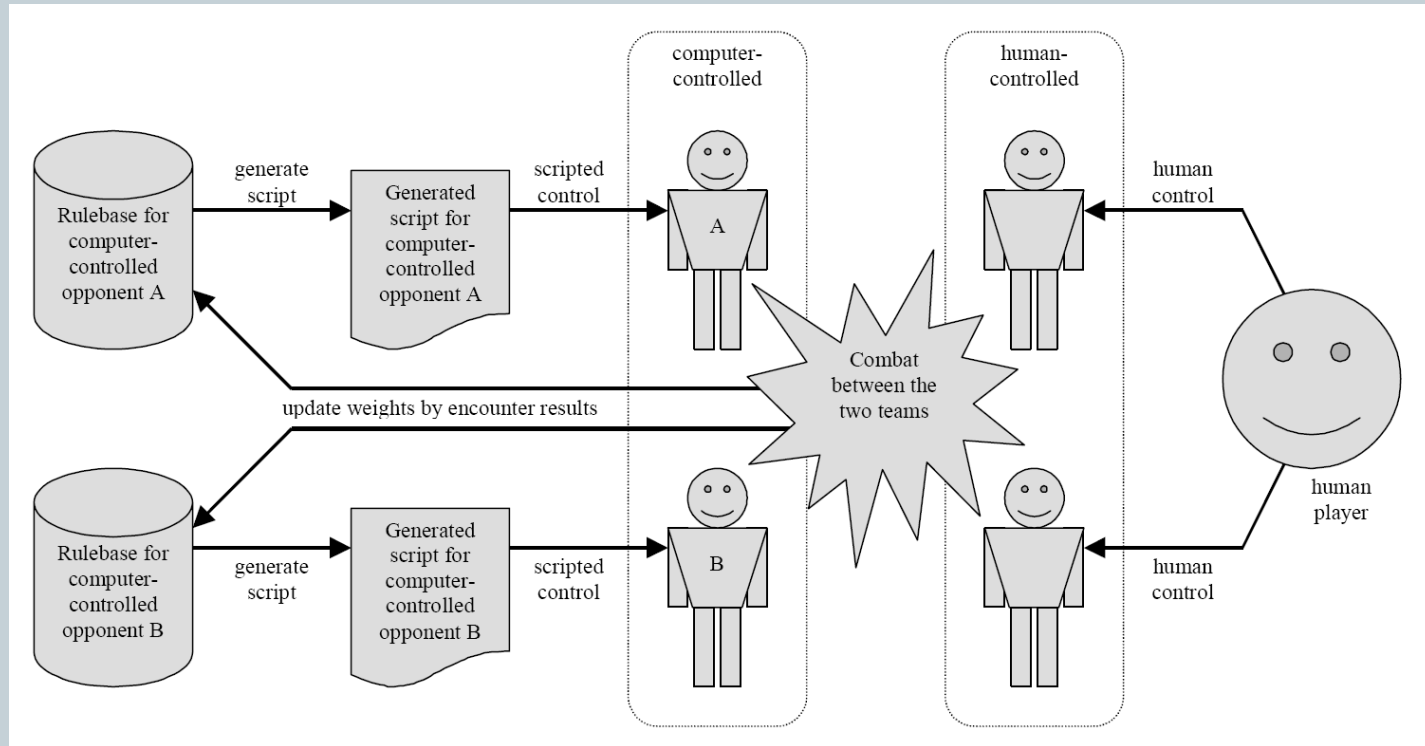


- **Problems with static scripts**
 - Fixed
 - Repetitive
 - Predictable
- **Project approach: reinforcement learning agents**
 - More fun to play (non static opponents)
 - More challenging (can exploit the human weaknesses)
 - Less error-prone (can fix its own design-time flaws)
 - More natural design standpoint

Introduction



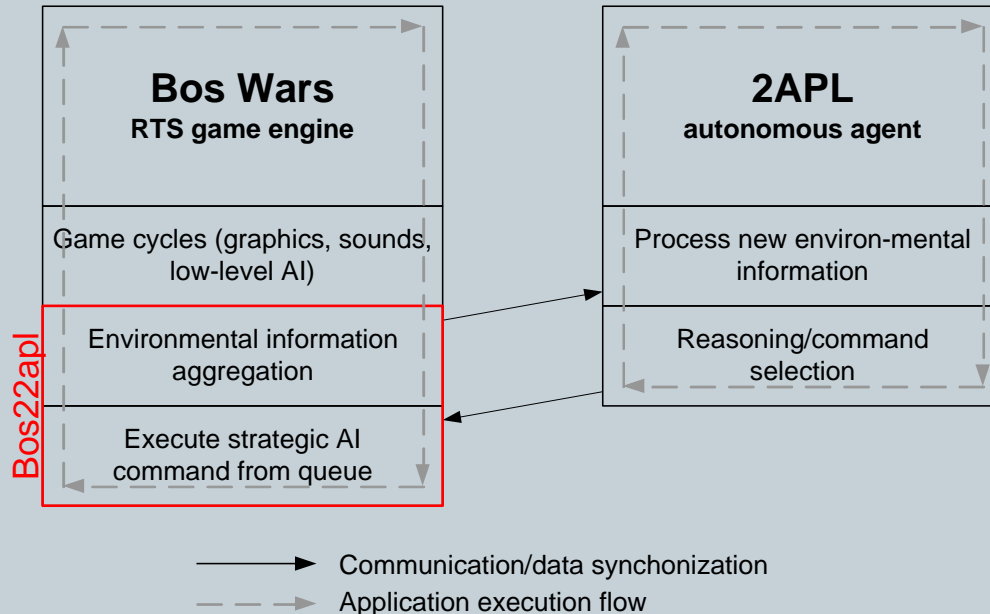
- Dynamic Scripting [Spronck *et. al.*, 2003]



Agents in Bos Wars



- Games and agents
- Benefits of 2apl agents
- Coupling 2apl to Bos Wars



Agents in Bos Wars



- Example: a simple Bos Wars playing 2apl agent

Beliefs:

```
gameCycle(o).
```

BeliefUpdates:

```
{ gameCycle(O) } UpdateCycle(N) { not gameCycle(O),  
gameCycle(N) }
```

Goals:

```
attack(enemy)
```

PG-rules:

```
attack(enemy) <-  
true | {  
prepare(base); }
```

PC-rules:

```
prepare(base) <-  
true | {  
@boswars(setUnit("unit-engineer", 4), R);  
@boswars(setUnit("unit-powerplant", 1), R);  
@boswars(setUnit("unit-magmapump", 2), R);  
@boswars(setUnit("unit-vfac", 1), R);  
@boswars(setUnit("unit-gturrent", 1), R);  
@boswars(waitUnit("unit-vfac"), R);  
launch(attack); }
```

```
launch(attack) <-  
true | {  
@boswars(defineForce(o, "unit-tank,4;"), R);  
@boswars(waitForce(o), R);  
@boswars(attackWithForce(o), R);  
dropGoal(attack(enemy)); }
```

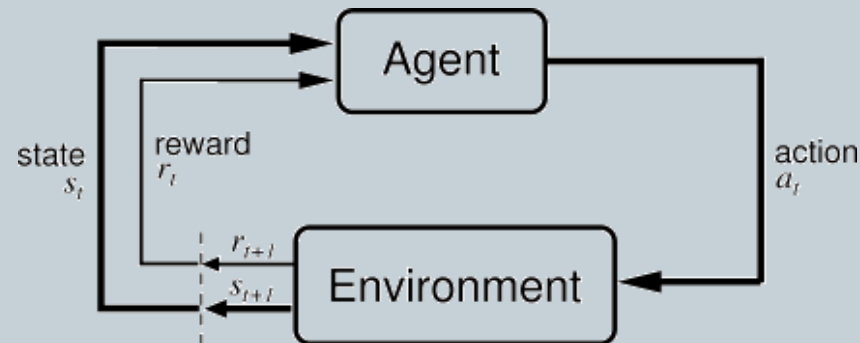
```
updateGameCycle(NewCycle) <-  
true | { UpdateCycle(NewCycle); }
```

```
updateGameResult(Result, Fitness) <-  
true | { @boswars(gameResultHandled(), R); }
```

Learning winning strategies



- Applying reinforcement learning to 2apl
 - Operating in an unknown environment
 - Action (PC-rule) selection over multiple weighted choices
 - Executing an (external) action returns a reward and a state
 - Update PC-rule weights to reflect obtained rewards



From Sutton, R. S. and Barto, A. G. 1998.

- Generic RL framework for Monte Carlo, TD, etc.

Learning winning strategies



- **Dynamic Scripting**
 - Tiny search space: Fast learning
 - Fixed script length
 - Rules may only be selected once
 - Fixed rule order
- **Monte Carlo control**
 - Classically has a large search space: Slower learning
 - No limitations on strategy
 - May use rule guards and a complex plan hierarchy

Learning winning strategies



• Example: a learning Bos Wars playing 2apl agent

Beliefs:

```
unit(unit-powerplant,o).  
unit(unit-camp,o).  
unit(unit-vfac,o).  
unit(unit-assault,o).  
unit(unit-tank,o).
```

Plans:

```
{ ^startEpisode(); ^setting("tau", 3); script(game); }
```

PC-rules:

```
^script(game) <- unit("unit-powerplant",X) and X < 1 | {  
  ^visit();  
  @boswars(setUnit("unit-powerplant", 1), R); }
```

```
^script(game) <- unit("unit-camp",X) and X < 1 | {  
  ^visit();  
  @boswars(setUnit("unit-camp", 1), R);  
  @boswars(waitUnit("unit-camp"), R); }
```

```
^script(game) <- unit("unit-vfac",X) and X < 1 | {  
  ^visit();  
  @boswars(setUnit("unit-vfac", 1), R);  
  @boswars(waitUnit("unit-vfac"), R); }
```

```
^script(game) <- unit("unit-camp",X) and X > 0 | {  
  ^visit();  
  @boswars(defineForce(o, "unit-assault,4;"), R);  
  @boswars(waitForce(o), R);  
  @boswars(attackWithForce(o), R); }
```

```
^script(game) <- unit("unit-fac",X) and X > 0 | {  
  ^visit();  
  @boswars(defineForce(1, "unit-tank,4;"), R);  
  @boswars(waitForce(1), R);  
  @boswars(attackWithForce(1), R); }
```

```
queueEmpty() <-  
true | { script(game); }
```

```
updateOwnUnit(UnitType, UnitCount) <-  
true | { UpdateOwnUnit(UnitType, UnitCount); }
```

```
updateGameResult(Result, Fitness) <-  
true | {  
  ^reward(Fitness);  
  ^endEpisode();  
  @boswars(gameResultHandled(), R); }
```

Learning winning strategies



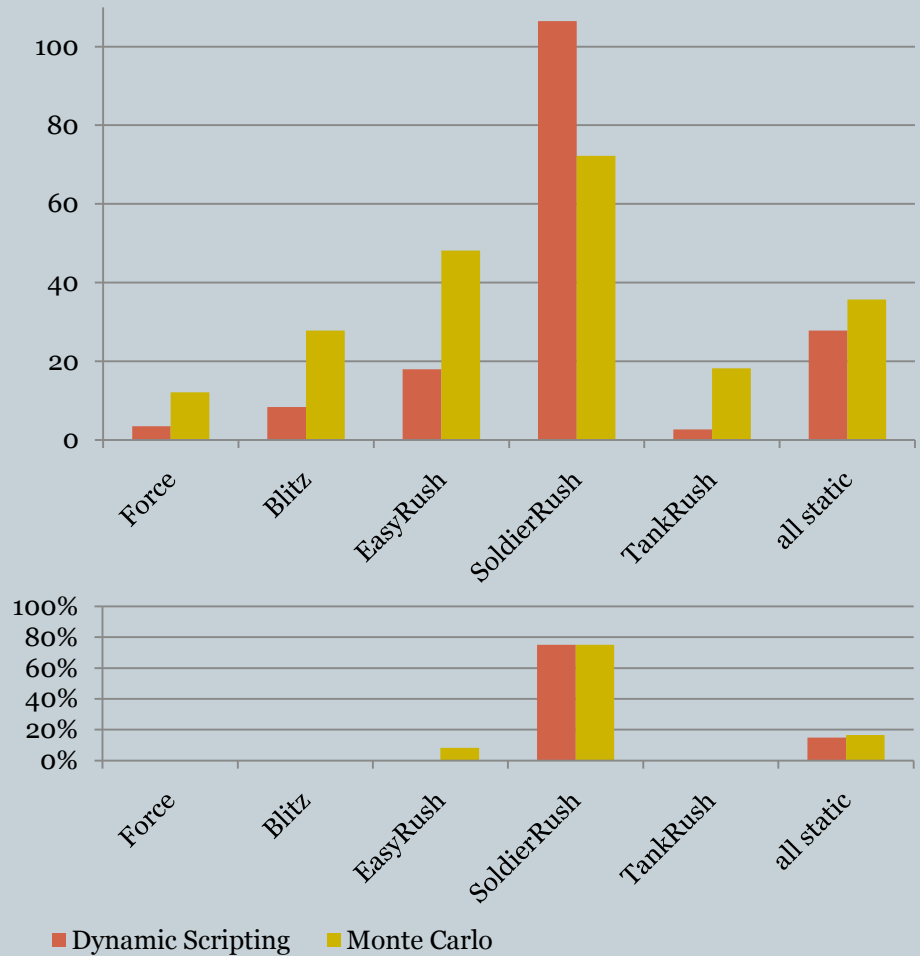
- **Experimentation setup**
 - Ran against 5 different static and 5 different strategy switching scripts
 - Game length of maximum 200.000 cycles (normally end within 100.000 cycles)
 - TP: Turning Point at which 12 points in 7 consecutive games are gathered (6 wins 1 lose or 5 wins 2 draws)
 - Fail: When no TP is reached within 200 episodes

Learning winning strategies



- Basic learning performance
- Monte Carlo uses
 - PC rule guards
 - Softmax exploration
- DS performs slightly better

Average Turning Points and Fail Rates



Learning winning strategies



- Improving an agent's learning performance
 - From flat, reactive agent to strategy hierarchy into PC-rules
 - ✦ Not effective in the short-paced strategy games of Bos Wars
 - ✦ When may this be more applicable?
 - MC-DS hybrid: use 'rule selection order' as game states

Adaptation to opponent tactics

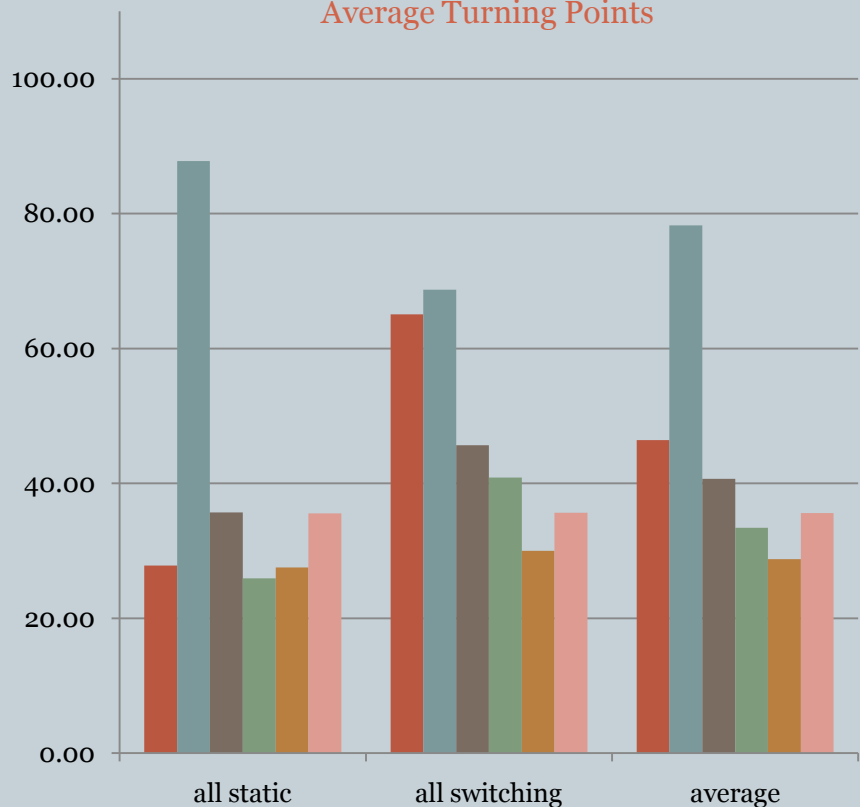


- **Opponent can switch strategies**
 - Switch when previous game was lost
 - Switch every 2, 4 or 8 games
- **Implicit adaptation through learning**
 - Incorporating opponent data in the game state
- **Explicit adaptation through strategy switching**
 - Learn distinct strategies using expected game result tracking

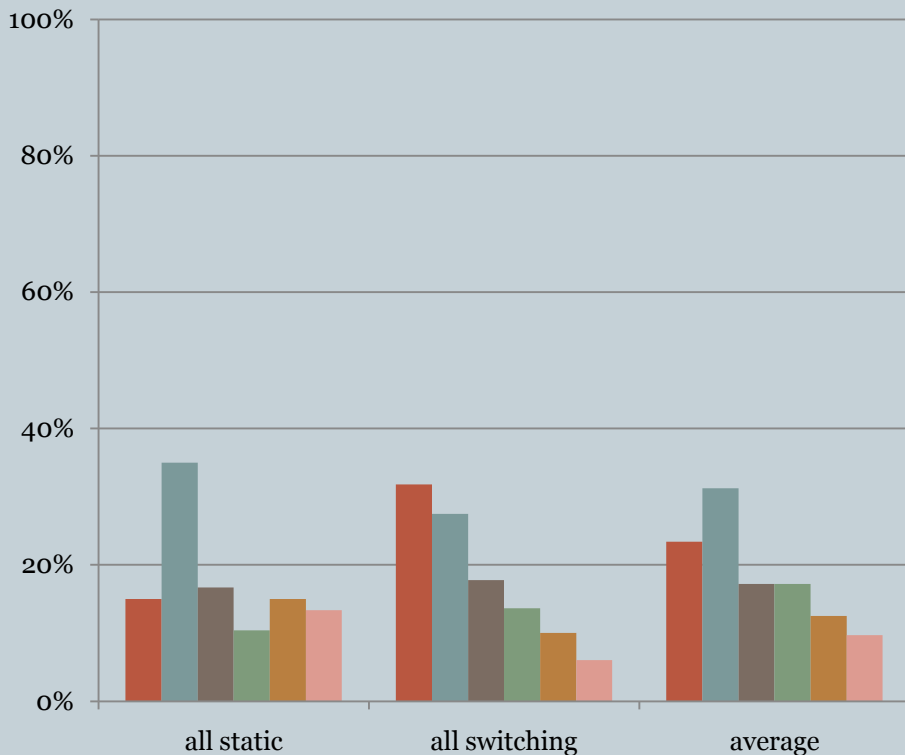
Experimentation results



Average Turning Points



Fail Rates



- Dynamic Scripting
- MC hierachy
- MC Softmax
- MC-DS Hybrid
- MC opponent data
- MC-DS Hybrid opponent data

Conclusions



- Agents are capable of learning winning strategies
 - More fun to play against
 - More challenging
 - Less error-prone
- MC can efficiently learn, needing less than 30 episodes on average
- Implicit adaptation with opponent data works best against strategy switching opponents
- MC-DS Hybrid may be used to replace complex game states, making DS redundant

References



- **Literature**

- [Spronck *et. al.*, 2003] Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma (2003). *Online Adaptation of Game Opponent AI in Simulation and in Practice*. Proceedings of the 4th International Conference on Intelligent Games and Simulation (eds. Quasim Mehdi and Norman Gough), pp. 93-100.