Cooperative Multi-agent Approach for Automated Computer Game Testing

Samira Shirzadehhajimahmood, I. S. W. B. Prasetya, **Mehdi Dastani**, Frank Dignum

https://iv4xr-project.eu/

Utrecht University

Automated Testing VS Manual Testing

Manual testing

Not reusable

Time and cost efficiency

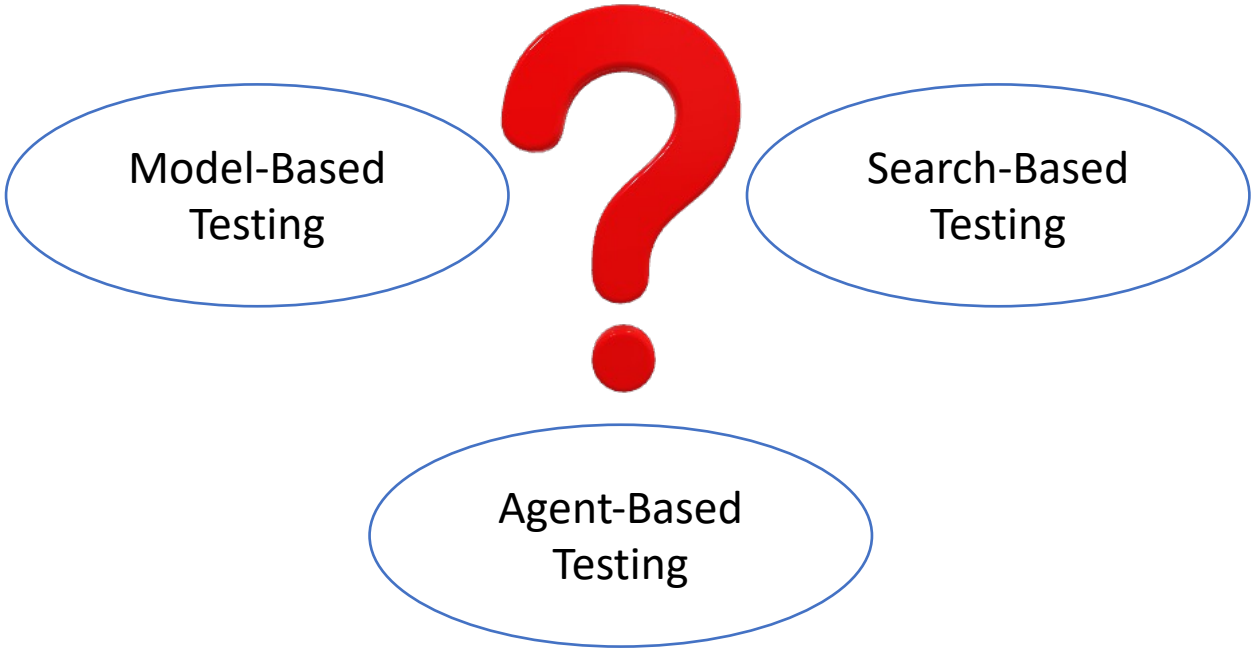
- https://www.capgemini.com/research/world-quality-report-2019/

Modern Computer Games

- Rich and complex environments
- Huge interaction space
- Multi Player
- Testing demands high human labor (User testing)

Challenges in Automated Games Testing

- The traditional ones are unable to handle the interaction space of 3D based Games
- Continues Space & Geometric Reasoning

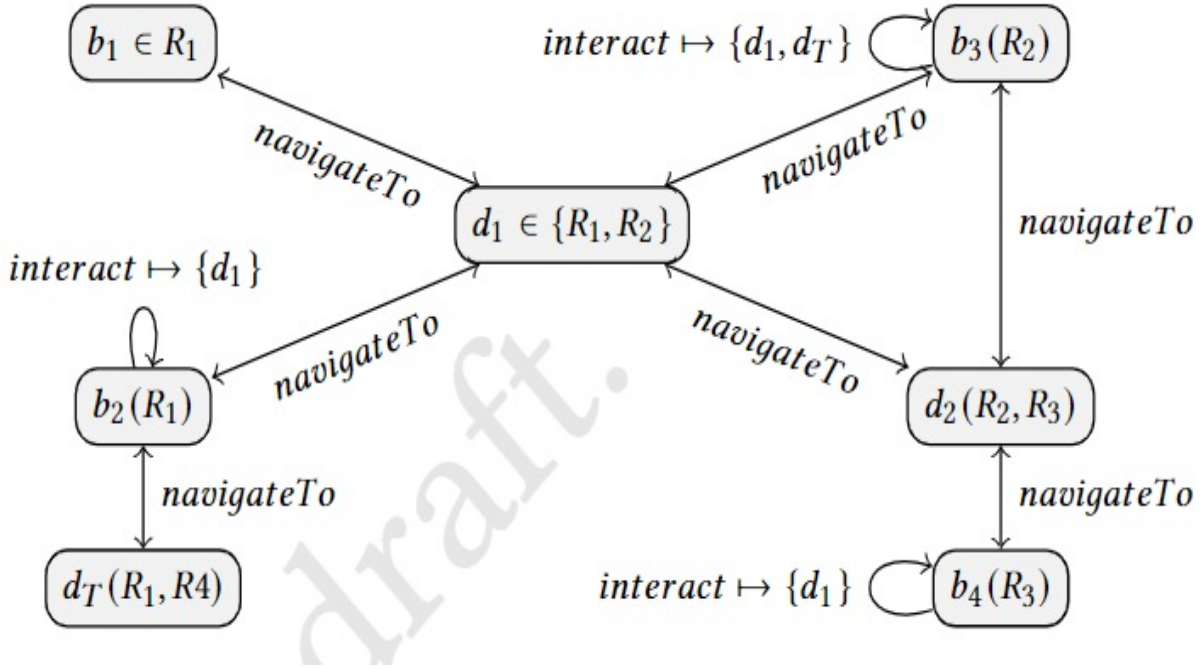


Agent-based automated game-testing (single agent)

Algorithm 1 Online Search

```
1:  procedure ONLINESEARCH(φ_o ⇒ ψ, M, Nav)
2:      while ¬φ_0 do
3:          parallel
4:              || update M_stategraph
5:              || if new states observed then
6:                  o' ← selectNode()
7:                  mark(o')                          ▷ mark it as visited
8:                  navigateTo(o') using Nav
9:                  if o' = o & ¬φ_o then
10:                     dynamicGoal(o', φ)
11:                 else if o' is a blocker & o.isblocked then
12:                     dynamicGoal(o', o'. ¬o'.isBlocked)
13:             else if there is terrain unexplored then explore()
14:             else abort()
15:             end if
16:         end parallel
17:     assert ψ
```

https://arxiv.org/pdf/2211.06936.pdf



An Online Agent-Based Search Approach in Automated Computer Game Testing with Model Construction, Shirzadehhajimahmood et al, ATEST 2022.

How about multi-agent game testing?

- **Motivation**: In the previous work, long test scenarios can take lots of time to run (e.g. 30+ minutes each). Many games are multi-player.
- **Hypothesis:** Deploying multiple agents can speed up testing (e.g. 0.5 reduction would be great!)
- **Challenges**: The agents can get in each others way, or even mess up each other plans. The computation overhead of just greedily sharing (and synchronizing) information may be too much.
- **Idea**: Collaboration between agents

Cooperative Multi-agent Approach

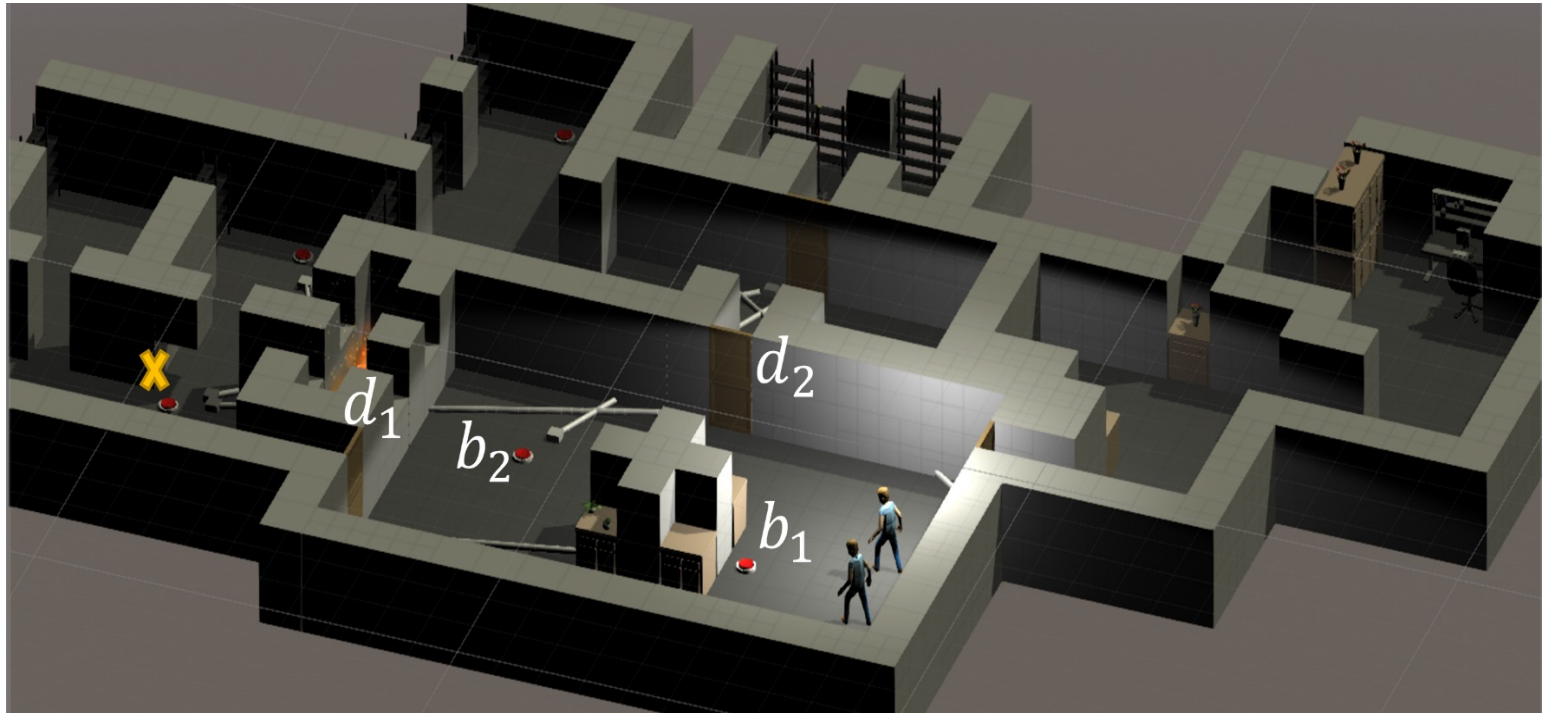
- ✓ **iv4XR framework**
  - Goal structure
  - Tactic

Agent-based Approach

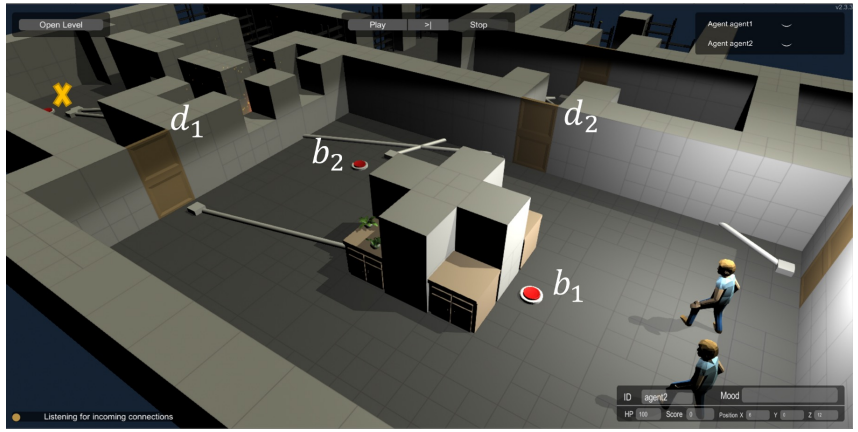
- Autonomy and reactivity
- Goal-based behavior
- Sensing and Decision Making

Multi-Agent Algorithm features

- Multiple test agents
- Working in parallel
- Dynamically choosing tasks
- Cooperative agents
- Dealing with obstacles



Multi Agent Testing Algorithm



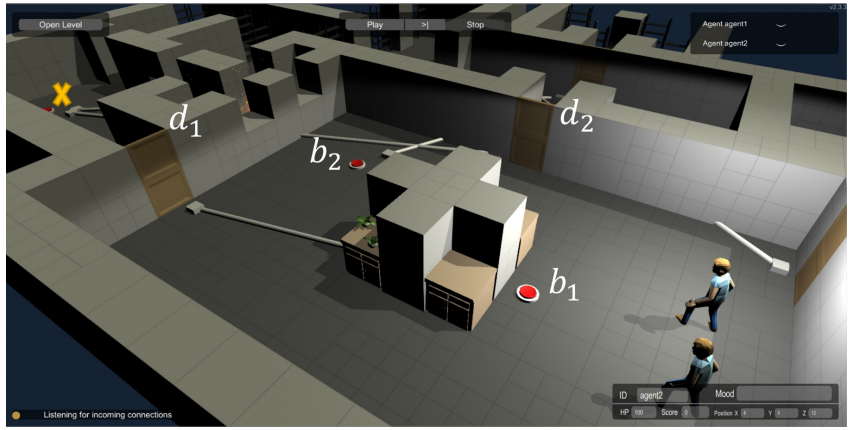
Algorithm 1 *It gets a set of tasks $\mathcal{T}$ and run N test agents.*

1: **procedure** COOPERATIVEAGENTS($\mathcal{T}$)
2: $toDo = \emptyset$
3: $done = \emptyset$
4: $agent_1$.SOLVER($\mathbf{H}_1$) $\parallel$ ... $\parallel$ $agent_N$.SOLVER($\mathbf{H}_N$)
5: $\parallel$ SYNC()
6: $\parallel$ $done{=}\mathcal{T} \vee budget \leq 0 \rightarrow$ **terminate** ▷ terminate the whole procedure

Algorithm 2 *For selecting and executing tasks, parameterized by two heuristics.*

1: **procedure** SOLVER($selectH, findH$)
2: **while** $budget > 0$ **do**
3: **if** $toDo \neq \emptyset$ **then**
4: $T \leftarrow selectH(toDo)$ ▷ use the task-selection heuristic
5: **if** $T \neq null$ **then**
6: $(o, \psi, S) \leftarrow T$
7: $toDo \leftarrow toDo/\{T\}$
8: NAVIGATETO(o)
9: **if** $\neg\psi$ **then** DYNAMICGOAL($o, \psi, S, findH$)
10: **if** ψ **then** $done \leftarrow done \cup \{T\}$
11: **else** $toDo \leftarrow toDo \cup \{T\}$
12: **else**
13: **if** there is terrain unexplored **then** FINDTASK( )
14: **else return**

Solving a chosen Task



Algorithm 3 *For solving a single task, parameterized by one heuristic.*

```
1: procedure DynamicGoal(o, ψ, S, findH)
2:     while ¬ψ ∧ ¬S do
3:         Δ ← {i | i ∈ seenObj ∧ ¬mark_o(i) ∧ ¬locked(i)}
4:         choose i ∈ Δ, based on findH          ▷ use the object-selection heuristic
5:         if i = null then
6:             if there is terrain unexplored then
7:                 EXPLORE( )                    ▷ explore world to find new objects
8:             else
9:                 return
10:        else
11:            mark_o(i) ← true                  ▷ mark i as tried for o
12:            LOCK(i)
13:            navigateTo(i)
14:            applyAction(i)                    ▷ such as interact
15:            navigateTo(o)
16:            UNLOCK(i)
```

Algorithm 4 *For finding new tasks.*

```
1: procedure findTask( )
2:     while there is terrain unexplored do
3:         basicExplore( )
4:         V ← newly observed objects
5:         W ← {(o, ψ, S) | o ∈ V ∧ (o, ψ, S) ∈ T/(toDo ∪ done)}
6:         if W ≠ ∅ then
7:             toDo ← toDo ∪ W
8:             return
```

Research questions

- RQ1: *does multi-agent speed up testing?*
- RQ2: *how well can multi-agent deal with complex logic?*

Information Synchronization Levels

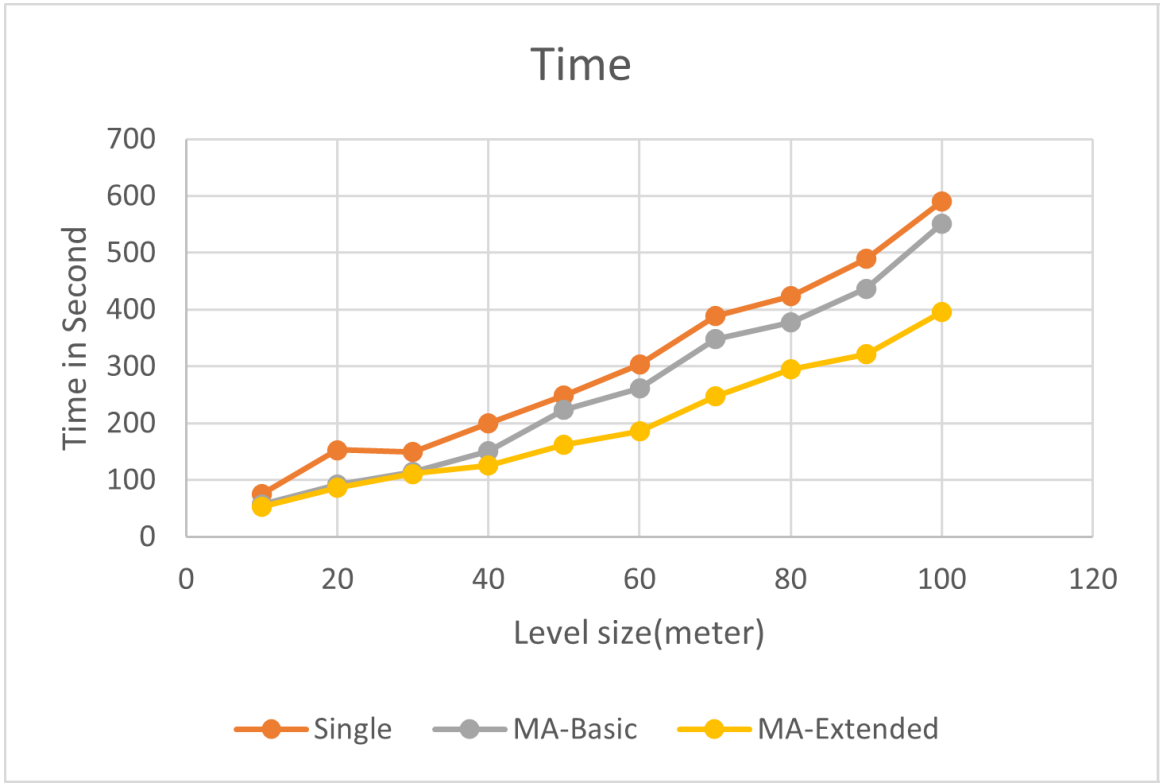
Basic: sharing seen tasks and solved tasks

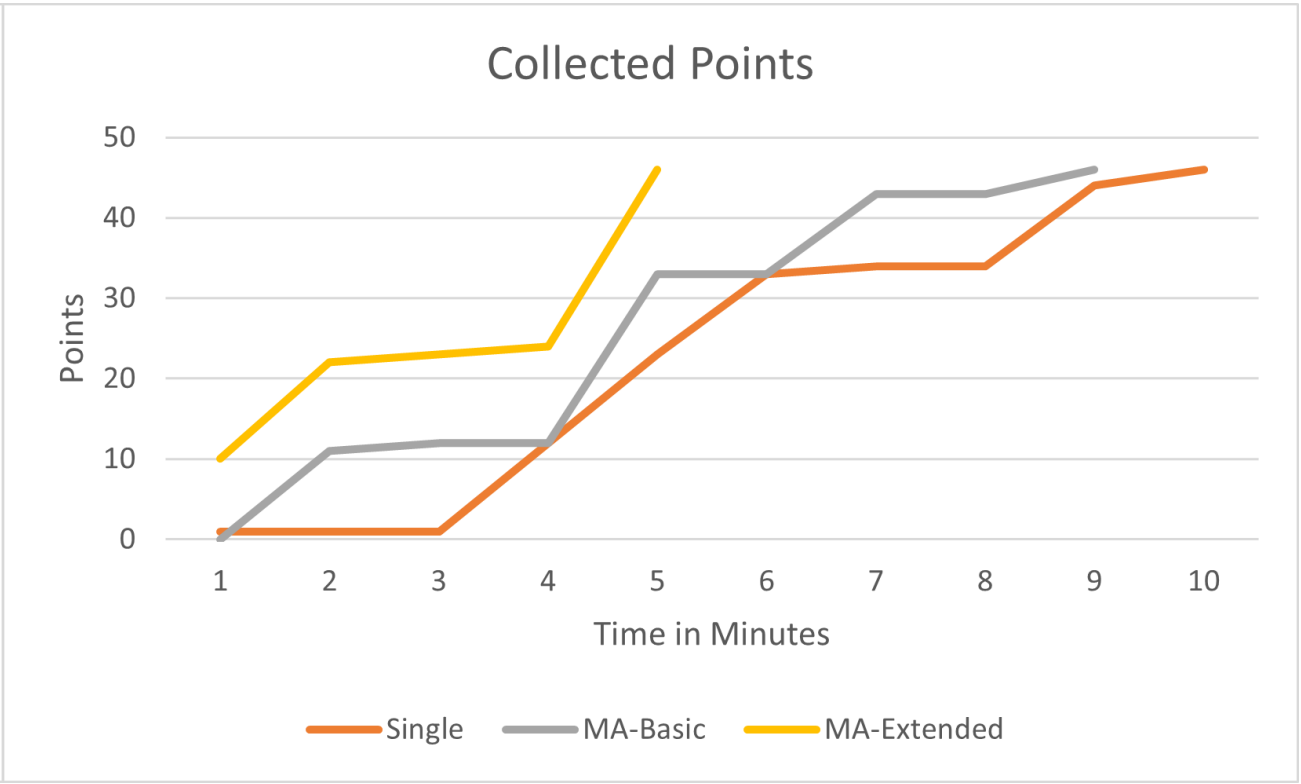
Advanced: Basic + sharing explored areas to each other

Factors Affecting the Performance

- Size of the game levels
- Task distribution among agents

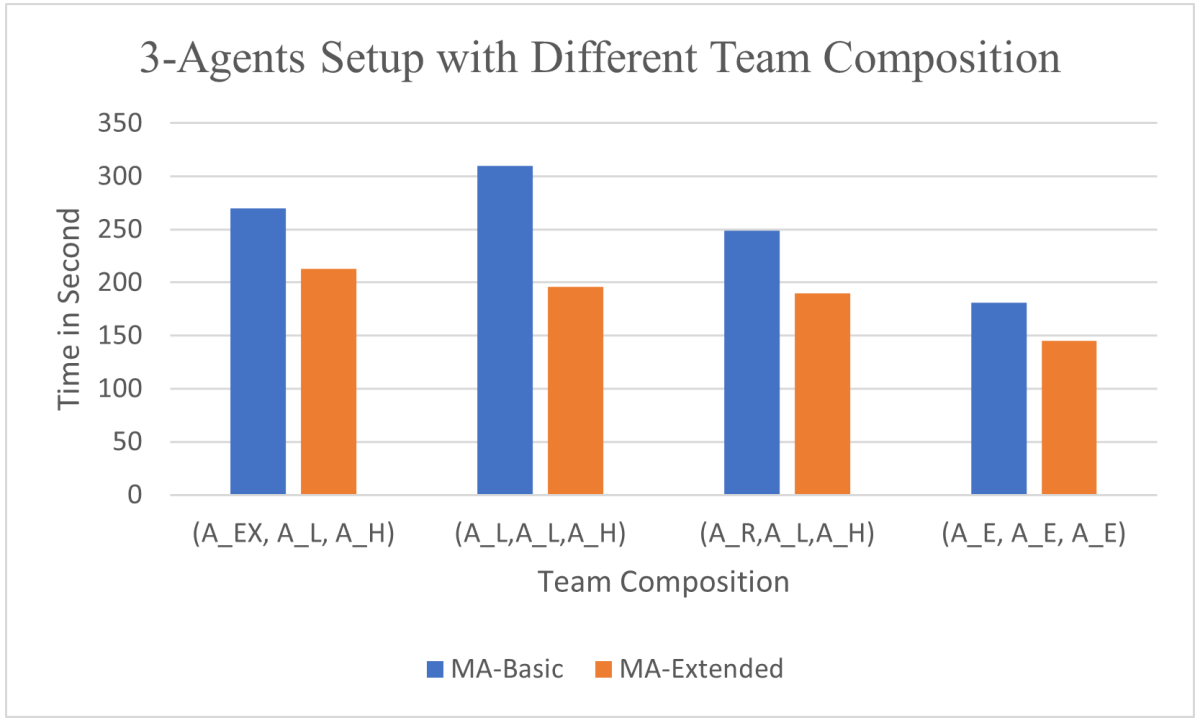
RQ1 and influence of information synchronization

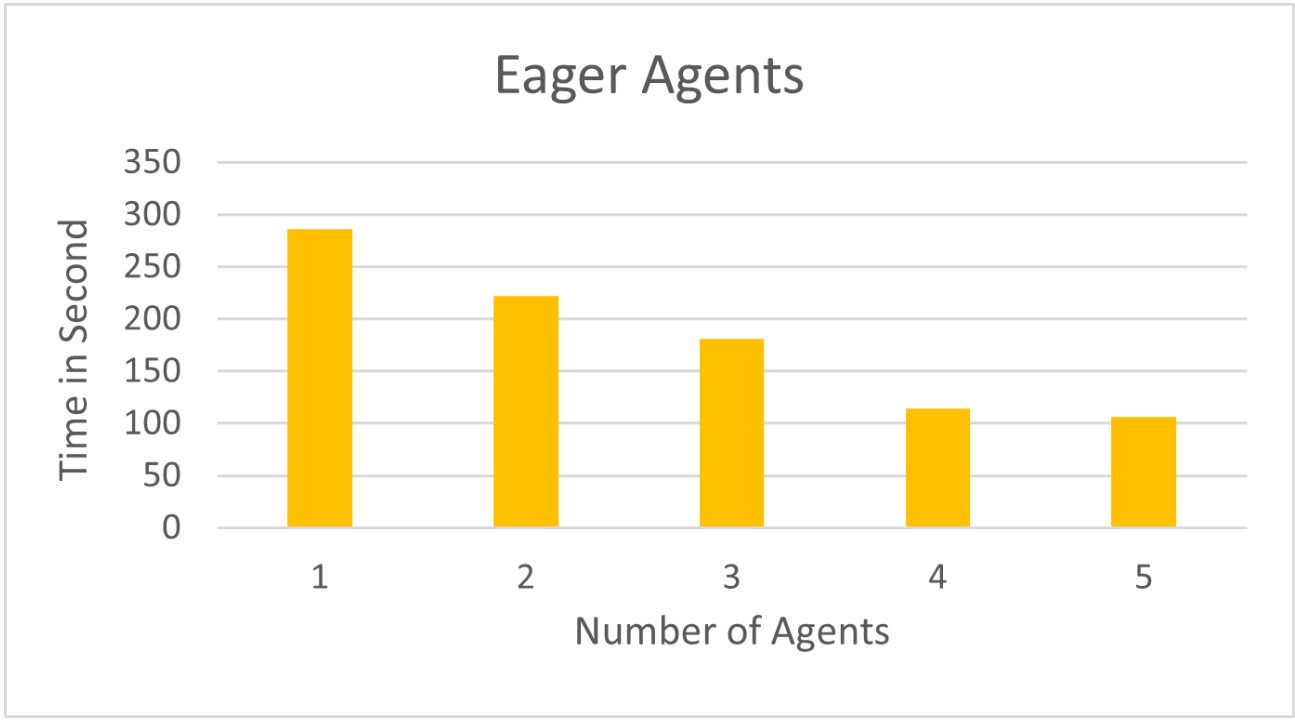




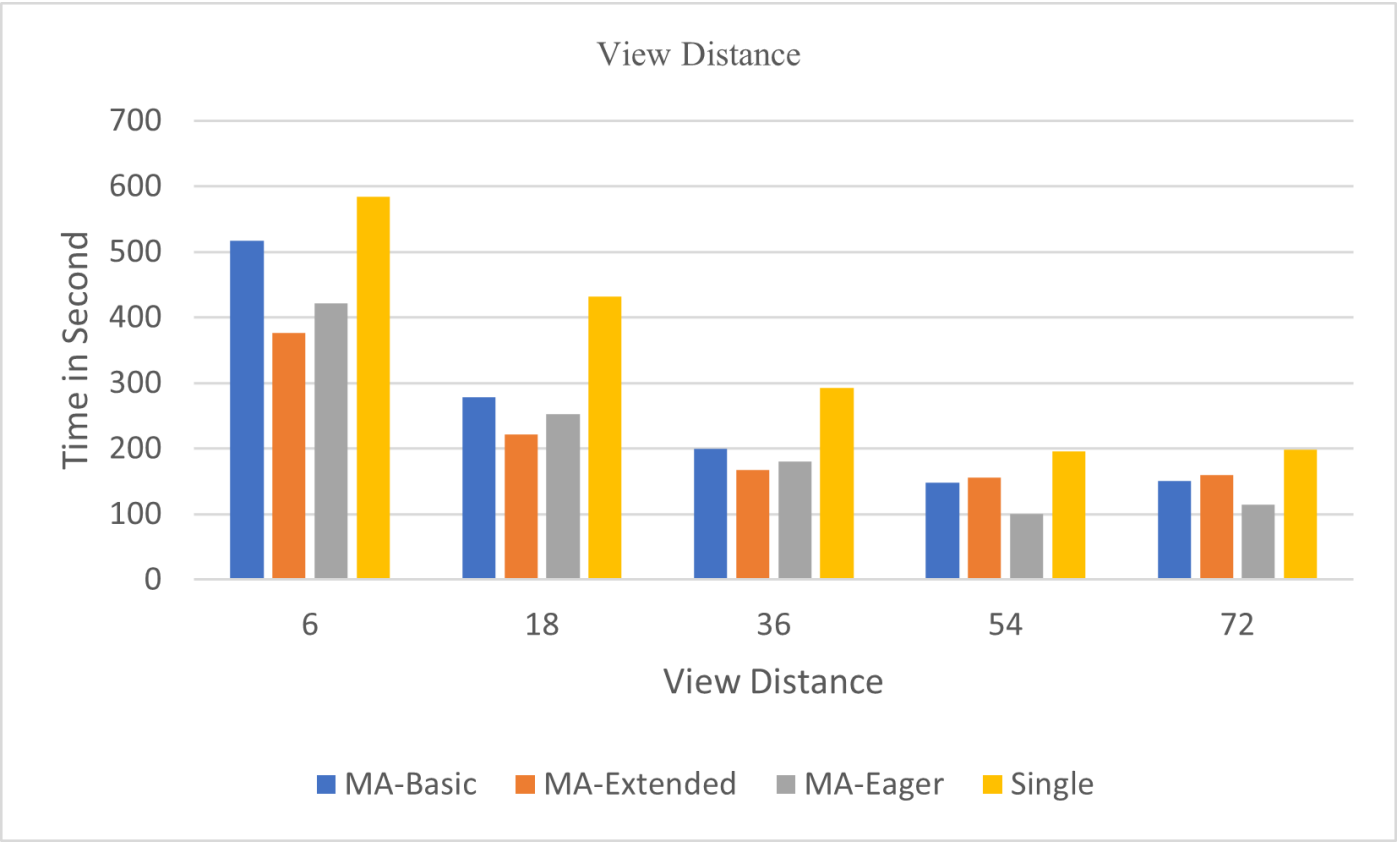
Points are gained when a goal is achieved

RQ1 and influence of team composition





RQ2: Distant-connection logic & hidden button logic



Distant-Connection

#DC	Single	MA-Basic	MA-Extended
2	210	159	117
4	265	209	195
6	315	289	215
8	352	333	232
10	180	352	264

Chained Connections

#HB	Single	MA-Eager	MA-Extended
0	149	128	111
1	214	145	140
2	307	254	191
3	337	308	241

Conclusion & Future Work

- ✓ A cooperative multi-agent testing approach

- ➢ Evaluated performance between a single-agent and multi-agent setups
  - Applying different heuristic to select tasks
  - Information sharing and synchronization

- ➢ Multi agent with extended information sharing performs better (despite the overhead).


Question?

iv4XR agent programming: tactic & goal Structure

A **tactic** is a way to hierarchically combine actions (or sub-tactics) to achieve a goal *G* :

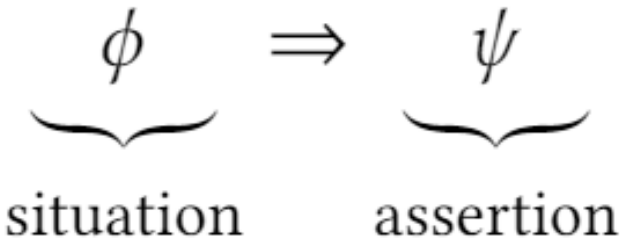
$$T = \mathbf{FIRSTof}(T_1, ..., T_n)$$

When executed, *T* runs in a loop over **multiple** deliberation cycles, until *G* is achieved (or aborted).

A **goal structure** is a set of goals that have to be achieved in a certain order:

$$H = \mathbf{SEQ}(G_1, ..., G_n)$$

Testing Task




```
var testingTask = SEQ(
        isAt(“button1”),
        isInteracted("button1"),
        isAt(”button3"),
        isInteracted(“button3"),
        …..
        isOpen(“treasureDoor”)
```

Sub-goals to solve/achieve

assertion

Domain Specific Language of iv4XR Framework

```
goal structure   ::=  SEQ(goal structure, goal structure,...)
                  |  FIRSTof(goal structure, goal structure,...)
                  |  WHILEDO(predicate, goal structure)
                  |  goal.lift()

goal             ::=  goal(name).toSolve(predicate).withTactic(tactic)

tactic           ::=  SEQ(tactic,tactic,...)
                  |  FIRSTof(tactic, tactic,...)
                  |  ANYof(tactic,tactic,...)
                  |  ABORT()
                  |  action.lift()

action           ::=  action(name).do(action expression).on(predicate)
                  |  action(name).addAfter(goalstructure)
```