Teamwork in Adversarial Video Games

Barbara Dunin-Kęplicz[1] and Rafał Tyl[2,3]

[1] Faculty of Mathematics, Informatics and Mechanics, Warsaw University
[2] Gentle Viking
[3] Grail

Abstract. In adversarial video games, human players and non-player characters (NPCs) compete to win certain goods or targets. To gain an advantage and create realistic behaviors, NPCs could cooperate by forming groups or coalitions. Surprisingly enough, they typically do not. Consequently, we have concentrated our efforts on providing game developers with the necessary tools to enhance virtual characters with teamwork capabilities. To start with, we address this challenge by observing that a classical Reconfiguration Algorithm, known in MAS, can serve as an umbrella for teamwork-related activities of NPCs. Reconfiguration amounts to intelligent replanning in response to changing circumstances. By analogy, our solution, inspired by the reconfiguration process, takes the control over the game and keeps it going by utilizing mechanisms that facilitate cooperation under adversarial conditions. As a proof of concept, we have developed, implemented, and tested ARA, the Adversarial Reconfiguration Algorithm, alongside an encompassing agent-oriented framework called ARAG and a simple stealth game called Treasure Hunt. ARAG facilitates a flexible approach to aspects of teamwork such as belief representation, communication, reconfiguration, and task allocation. As demonstrated by Treasure Hunt, NPCs coordinated by ARAG outperform those lacking cooperative capabilities, even with the adopted simplifying design choices.

Keywords: methodology of teamwork, reconfiguration algorithm, adversarial video games

1 Introduction

Teamwork matters. This paradigmatic statement is true in multiagent systems. But how about video games? Most games include non-player characters, NPCs, who engage human players by providing adequate challenges and exhibiting convincing behaviors [25]. In adversarial video games, opposing parties compete to win certain goods or targets. NPC teammates could work together to gain an advantage, but despite recent attempts to improve NPCs cooperation with basic communication and coordination [1], this is an often neglected aspect of video games [2,25]. The state of the art motivated us to investigate how to automatically orchestrate NPCs' teamwork in adversarial video games.

Teamwork, typically regarded in the literature as a collective accomplishment of common goals [6,28,26,27,5,19], can take various forms [15]. Traditionally, the study of cooperation belongs to MAS, so we searched for a relevant methodology there, choosing TeamLog, a formal BDI model of agents' cooperation introduced in [5]. Based on multi-modal logic, TeamLog, has been designed to model group attitudes like collective intentions and collective commitments during teamwork. Their role is highlighted in the classical model of teamwork, consisting of stages of potential recognition, team formation and plan generation, culminating in team action. Namely, a collective intention constitutes a strictly cooperative team during team formation, while a plan-based collective commitment, implementing the specifics required for a group to cooperate, is set up during planning. Ultimately, reconfiguration, as a form of intelligent replanning, is carried out throughout team actions and implemented as a Reconfiguration Algorithm [5].

For several reasons, we are hesitant to use TeamLog directly, even though it would be natural to consider it as the underlying teamwork modeling formalism. First, while situated at the core of TeamLog, complete BDI agents with groups equipped with collective attitudes are not really needed at the current stage. Second, by turning the player into an opponent, the video game generates an entirely distinct situation. Specifically, the NPCs' team must make a clear distinction between the effects of the enemy's actions and those arising from the environment's variability. Third, industry standards of game specification have to be taken into account. All in all, we do not adhere to the logic-based formalism of TeamLog, while still remaining inspired by its methodology.

Taking this into account, we developed the Adversarial Reconfiguration Algorithm (ARA) to dynamically manage team actions in response to game state changes. It draws from MAS-related Reconfiguration Algorithm and AI planning, providing an orchestration of NPC behavior in adversarial video games. The entire process is implemented as the Adversarial Reconfiguration Algorithm for Games (ARAG) framework, automatically integrating teamwork into games.

Although the burden of game design and implementation remains with the developers, ARAG offers the advantage of fully implementing teamwork among otherwise uncooperative NPCs. As an input, a developer has to provide an implementation of an adversarial game without collaborative elements along with components required by ARAG, via a standardized API. As a result, ARAG produces a realistic teamwork of NPCs during the course of the game.

Compared to ad hoc solutions, ARAG offers multiple qualitatively innovative game possibilities, combining MAS methodologies with game industry experience. One of the challenges in agent-based programming is a combination of goal-directed reasoning resulting from planning with data-driven stream reflecting the environmental circumstances. Even though game environments are usually simpler than those in MAS, one must still account for unanticipated events brought on by human players. ARAG addresses this issue.

Overall, the original contributions of this paper include:

- ARA: the algorithm for managing team actions in reaction to changes,
- ARAG: the framework for orchestrating NPCs' teamwork, encompassing ARA,
- a prototype implementation for a simple stealth game - Treasure Hunt.

The paper is structured as follows. In Section 2, we discuss teamwork in adversarial video games, while Section 3 describes ARAG's role in coordinating teamwork. Section 4 is devoted to the ARAG architecture. Section 5 presents the core of our research: ARA with the auxiliary procedures. In Section 6 we discuss the design choices underlying ARAG, while Section 7 presents a proof of concept. Section 8 discusses related work and, Section 9 concludes the paper.

2 Teamwork and Adversarial Video Games

Video games are complex systems, where an audiovisual apparatus is orchestrated by game engines facilitating the development of components like resource management, rendering, game loop management, physics simulation, etc. [7]. Thanks to the availability of affordable game engines [18], developers do not have to build games from scratch, but provide game-specific components solely. A perception of the game depends on the level of realism in each individual action and on the complexity of the NPCs' interactions. Moreover, cooperative teams of NPCs must battle against (human) player and be able to:

- operate in a physical environment with interactive elements (e.g. doors),
- coordinate actions with others,
- access the game world data,
- adequately react to adversarial actions of others,
- communicate beliefs or actions by graphical symbols (icons and indicators) or sounds (so called "barks").

In multi-agent environments, actions may fail due to unexpected circumstances. In adversarial video games, one needs to distinguish situations resulting from a natural variability of the environment from those caused by an enemy. This requires a proper separation of:

- observed phenomena,
- messages explicitly/implicitly dealing with adversarial acts,
- a lack of contact with some agents.

Video games typically progress according to scenarios scriptwrited by game designers. These scenarios outline the principles governing dynamic alterations to the surroundings, such as those that occur naturally (like an accident), on a regular basis (like day-night cycle mechanics), or as a result of player or NPCs activity (like opening and closing a door). Thus, a key concern is how to strike a balance between a designer's control and the autonomy of NPCs. The adherence to scripts is ensured by plans applicable in specific circumstances: individual plans, reflecting individual behavior and social ones, reflecting team behavior. Both types are constructed from actions to be performed by particular NPCs in a given order. Thus, ARAG utilizes a user-defined plan library as the foundation of teamwork, while still being able to enhance the game with complex and diverse teamwork functionalities.

During team action, a hostile interruption of a plan's execution necessitates an appropriate response. Although plan adaptation is prevalent in MAS, in games a team must explicitly monitor individuals for prospective attacks. Thus, replanning during reconfiguration influences both group and adversary behavior. These aspects are implemented in ARA.

3 ARAG as a teamwork coordinator

In video games, most of the time NPCs fight human players. The idea of ARAG is to give NPCs the opportunity to cooperate in order to increase their gains and make the task more difficult for their opponents. To achieve this, ARAG serves as a coordinator of NPCs' teamwork, while maintaining the individual roles assigned to them by the game designer.

3.1 ARAG and NPCs

ARAG takes control over NPCs in the game. Functionally, it can be thought of as a software agent whose responsibility is to manage game-specific behaviors of NPC agents. As its input, ARAG receives an implementation of an adversarial game. As usual, the game is created and implemented by the game creator. The use of ARAG adds an additional overhead, including some situation-specific preferences and game-related strategies (listed in Section 4.3) that need to be made available via an ARAG-imposed API. As the result, ARAG enhances the game by automatically created realistic teamwork of autonomous NPCs.

The ARAG's operations, encapsulated in ARA, lead to the creation, execution, and intelligent modification of social plans, accomplished by techniques derived from distributed AI and MAS. Consecutive phases of the entire process allow to change a plan or a team, according to the hazards of the game.

NPCs and ARAG maintain some explicit representation of beliefs, commitments and reports, but we are not concerned with their specific representations. As regards beliefs, we assume that there is an interface to the belief base providing fused beliefs of all NPCs, abstracting from specific belief fusion methods. They may vary from the simplest belief union to sophisticated fusion methods, e.g. eliminating noise or resolving potential uncertainty, inconsistencies and/or disagreements. In contrast to real-world applications, in video games NPCs typically have a direct access to the virtual environment. Thus their observations can be interpreted unambiguously so a simple sum of beliefs is often sufficient.

3.2 Plan skeletons and social plans

ARAG realizes the two main aspects of teamwork – task sharing and result sharing [27]. Task sharing involves the decomposition of a comprehensive task into subtasks assigned to individual agents. These tasks form social plans that transform a loosely connected NPCs into a cooperative team. Result sharing pertains to supplying teammates with relevant information.

The social plan construction pertains to a user-provided library of plan skeletons, defined as a partially ordered set of tasks, in particular denoting tasks to be performed sequentially. A complete social plan is a defined as a partially ordered set of task-agent pairs, where each pair denotes an agent allocated to a specific task [5]. A transition from a plan skeleton to a social plan is shown in Figure 1.

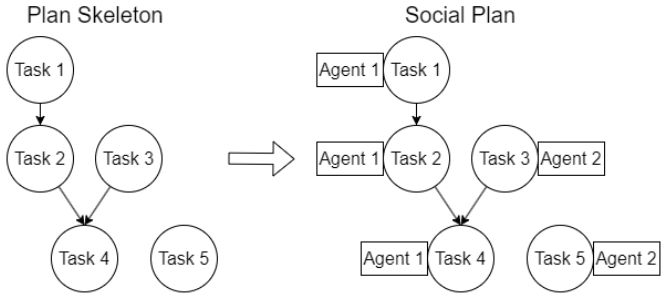

Fig. 1: Transformation of plan skeleton into a social plan.

The ARAG activity starts from team formation and task allocation, resulting in a social plan. Ultimately it leads to a team action, that is, to collective execution of this plan by a team of NPCs under ARAG's supervision. During plan execution, ARAG determines the status (*in progress*, *completed*, or *failed*) of the plan as a whole. If a task fails, ARAG reconfigures the plan, as described in the sequel. In short, ARAG is responsible for:

- choosing the plan to achieve the objective,
- identifying the most suitable NPCs,
- creating a cooperating team,
- developing a social plan,
- distributing tasks to NPCs,
- creating bilateral commitments between NPCs and itself,
- supervising a plan execution and reconfiguring it,
- changing the plan in the case of emergency/lost connection,
- dissolving the team, when needed,
- announcing failure/success/abort of the plan.

Commitments between ARAG and NPCs are created by ARAG sending a task execution request and the NPC confirming its ability to perform this task. Then NPCs carry out tasks and report to ARAG their progress and observations pertinent to the team. This way ARAG becomes the central node of the commitment network. Such a star topology centralizes some agent management aspects while reducing the need for group interactions.

An NPC is primarily responsible for carrying out agent-specific tasks. What counts to ARAG is whether the NPC is capable of performing the tasks, as indicated by its opportunities and abilities reflecting its objective possibilities and subjective skills, respectively [12,9]. That is, NPCs are heterogeneous.

4 The Architecture of ARAG

ARAG consists of multiple interacting components and databases (Figure 2):

- *Belief Base* of beliefs about the world and NPCs;
- *Report Base* of task status reports,
- *Plan Library* of plan skeletons.

Information stored in the databases (denoted as ovals) is processed in order to create, monitor and reconfigure social plans by the following processes (denoted as rectangles):

- *Reasoner*, responsible for reasoning about beliefs,
- *Opportunity Analyzer*, producing feasible tasks, based on the current beliefs about the environment,
- *Report Analyzer*, checking the current plan's status and generating new tasks based on monitoring results,
- *Filters: Compatibility Filter, Task Override Filter*, narrowing down the stream of tasks,
- *Task Selector*, selecting the most suitable task,
- *Means-End Reasoner*, selecting applicable plans from Plan Library,
- *Adversarial Reconfiguration Algorithm* ARA, creating the details of a *social plan* and reconfiguring it, when necessary,
- *Communication Layer*, facilitating communication between ARAG and NPCs.

In contrast to Reasoner and Means-End Reasoner that are adequate to the representation of beliefs and plans, Filters, Task Selector, Report Analyzer and Opportunity Analyzer are game-specific and their content should be delivered by the user (e.g., as a set of rules).

The communication layer serves as the interface between ARAG and the team, ensuring effective information exchange. Even though MAS are known for complex interactions, in ARAG, designed for real-time video games, they should be as simple and balanced as possible. Therefore, only the most fundamental speech acts — a bilateral communication between two actors and an announcement, when the message reaches all teammates simultaneously — are considered. Facilities ensuring proper communication are a part of ARAG, while the potential presentation layer (sounds, animations etc.) should be delivered by the user.

4.1 Combining Various Streams of Options

Adversarial Reconfiguration Algorithm refers to optional tasks originating from:

- the current plan: goal-directed aspect,
- the observed/communicated environmental changes: data driven aspect,
- the monitoring process: potentially generating new options.

Given the recent circumstances, all tasks originating from the above three sources compete to become the winner - the task to be executed next. This issue is central in intelligent decision systems and may be solved in a variety of ways. Based on the experience acquired in MAS, a filtering system inspired by a well known system IRMA [3] was designed in a relatively transparent way. As filters aim to reduce computational complexity during task selection, their sensitivity is crucial: too sensitive might destabilize the ARAG's operation, while too rough might downplay potential threats [3].

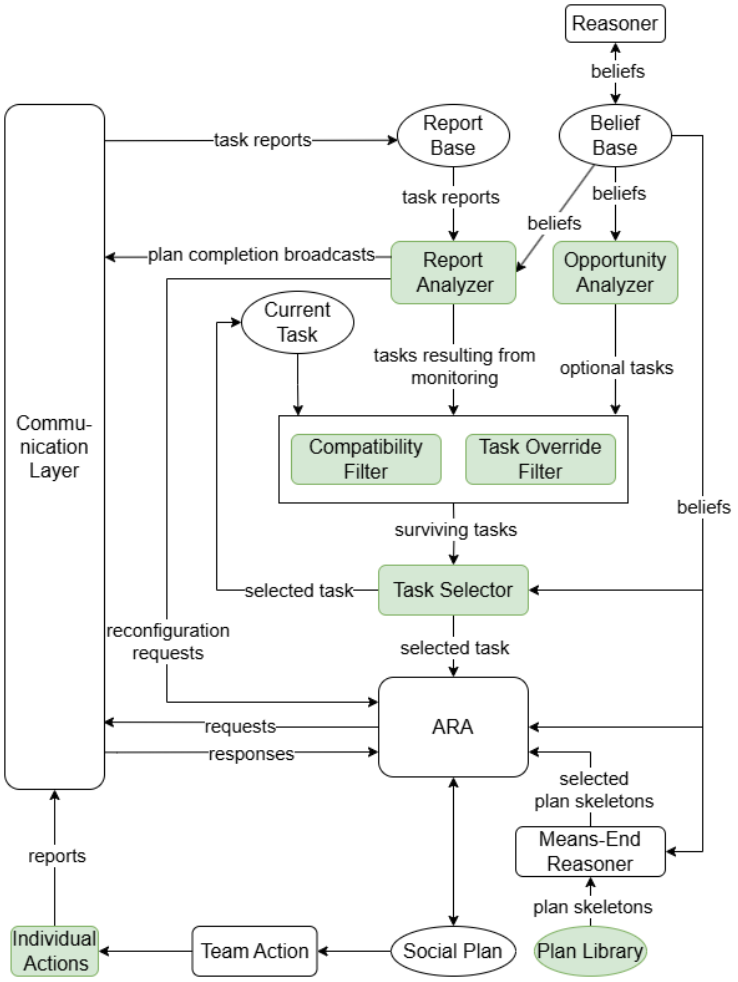

Fig. 2: ARAG architecture

4.2 The ARAG's Lifecycle

ARAG is incorporated into the game as a software framework, meaning that the user provides required data and procedures for calculating game-specific parameters (like abilities and opportunities), as outlined in Section 4.3. On the other hand, ARAG dictates the flow of control inside its update function called periodically from the game loop. Namely, on each update the lifecycle of ARAG starts with generating three streams of optional tasks, then sent to the Compatibility Filter to assess their consistency with the current task. Importantly, the rejected options are not simply discarded. They pass through the Task Override Filter to determine whether the current situation requires a radical change of the plan. In such cases, for example, after detecting hazardous conditions, relevant options are passed to the Task Selector. Out of surviving tasks, the Task Selector chooses the best one to pursue. A relevant social plan is formed using Adversarial Reconfiguration Algorithm, described in Section 5. Whenever the system is updated, the plan is modified or the task is modified in response to environmental changes.

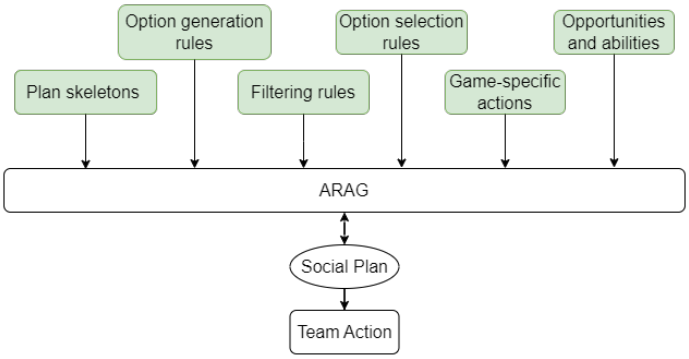

Fig. 3: ARAG inputs and outputs.

4.3 ARAG's Setup Step by Step

The user is required to provide the following game-specific elements:

- plan skeletons for the Plan Library,
- filtering rules,
- option generation rules for the Report Analyzer,
- option generation rules for the Opportunity Analyzer,
- option selection rules for the Task Selector,
- game-specific actions of the NPCs,
- functions to compute individual opportunities and abilities.

In Figure 2, the modules containing user-provided data are indicated in green. They are user-tailored and establish the level of detail of the team's interactions. The rules from analyzers, together with filtering and task selecting mechanisms, ensure a deliberative aspect of ARAG, leading to the choice of the next task for the team. On the other hand, user-provided plan skeletons enable scenario-driven means-end reasoning as well as event-driven reactions to both environmental changes and enemy activity.

As a core of teamwork, ARAG produces a social plan to achieve the current task by NPCs. Then it monitors the execution of the resulting team action. Specifically, this social plan and coordinated team action constitute the ARAG 's output. A higher level view of inputs and outputs is shown in Figure 3.

5 Adversarial Reconfiguration Algorithm

The Adversarial Reconfiguration Algorithm, ARA, provides the overall control structure of a team. It applies practical reasoning to create social plans. Further plan modifications ensuring effective execution of the resulting team action are accomplished by separating successive phases of the algorithm and controlling their complex interplay. In contrast to the classical Reconfiguration Algorithm [5], ARA consists of five phases, differing from the original ones. In each phase, the algorithm moves towards creating a team that executes a social plan, possibly returning to an earlier phase in the case of failure. These phases are implemented in pseudocode as:

1. plan selection - in Algorithm 1,
2. team formation - in Algorithm 1,
3. task allocation - Algorithm 2,
4. commitment distribution - Algorithm 3,
5. team action monitoring - Algorithm 4.

To avoid nested loops and improve readability we use 'go to' statements.

5.1 Plan Selection

Planning boils down to taking the next plan skeleton from the sequence of applicable plans, returned by the Means-End Reasoner and sorted by preferences. Each plan contains information about the optimal size of the team that will be consumed during team formation. The entire procedure fails in the absence of a plan, indicating that the task is not feasible given the current situation and the pool of potential NPCs.

5.2 Team Formation

Once a plan is selected, unsuitable NPCs are eliminated, based on their abilities and opportunities for specific tasks. Then, Payne's revolving door algorithm [10] allows to choose subgroups of the necessary size by selecting k-subsets that differ by only one member. These subgroups are successively passed to task allocation. If all accessible teams have been tried and failed, or if there are not enough NPCs, the algorithm returns to planning to select an alternative plan.

Algorithm 1: ARA

Input: $task$ - the current task
Input: $potentialTeammates$ - the list of potential teammates
Input: $planToReconfigure$ - the social plan to be reconfigured. If null, a new social plan will be created.
Data: $beliefBase$ - the belief base
Output: a social plan realizing the current task

```
/* if the current plan is reconfigured, start with team formation first            */
if planToReconfigure ≠ null then
    plan ← planToReconfigure
    go to TEAM_FORMATION;
end
plans ← queue of plan skeletons from Means-End Reasoner suitable for task
PLANNING:
/* if the algorithm returned to planning after a failed reconfiguration, start over
   */
if planToReconfigure ≠ null then
    plans ← queue of plan skeletons suitable for task from Means-End Reasoner
    planToReconfigure ← null
end
if plans is empty then
    return null; // fail and stop the algorithm
end
plan ← dequeue the first element of plans
teamSize ← team size from plan's metadata
/* team index is reset after getting a new plan, to be used during team formation  */
currentTeamIndex ← −1

TEAM_FORMATION:
consideredAgents ← filter potentialTeammates by ability
if size of consideredAgents < teamSize then
    go to PLANNING
end
/* produce the next team of the given size using Revolving Door algorithm          */
increment currentTeamIndex
team ← RevolvingDoor(consideredAgents, teamSize, currentTeamIndex)
/* team formation fails if there are no more teams of the given size               */
if failed then
    go to PLANNING
end

TASK_ALLOCATION:
globalContext ← construct global context based on beliefs from beliefBase
socialPlan ← Allocate(plan, team, globalContext)
if failed then
    go to TEAM_FORMATION
end

COMMITMENT_DISTRIBUTION:
DistributeCommitments(socialPlan)
if failed then
    clean up commitments
    go to TEAM_FORMATION
end
return socialPlan
```

5.3 Task Allocation

The allocation phase assigns teammates to tasks from the plan skeleton. For each task ready for execution, the most suitable NPC is selected by computing a quality score based on opportunity and ability scores. If no NPC achieves a score above a user-specified threshold, the algorithm reverts to team formation. The specifics of this process are outlined in Algorithm 2. The entire procedure repeats until all tasks are assigned, completing the social plan.

Algorithm 2: Allocate

```
Input: plan - a plan skeleton with NPCs not assigned to tasks
Input: team - the team that should execute the plan
Input: globalContext - a copy of a relevant subset of the belief base that is used as
       "working memory" during allocation
Data: minScore - the minimal required allocation score set by the user
Output: social plan

socialPlan ← plan with empty allocation slots
unallocatedTasks ← all tasks from plan
/* tasks ready for allocation are determined, as described in Section 5.3          */
tasksReadyToAllocate ← ready tasks from unallocated tasks
// proceed until all tasks are allocated
while size of unallocatedTasks > 0 do
    if tasksReadyToAllocate is empty then
        // the plan is malformed
        error
    end

    bestAgent ← null
    bestTask ← null
    highestScore ← 0
    /* go through the possible allocations and select the best one, based on
       opportunity and ability scores                                              */
    for task ∈ tasksReadyToAllocate do
        for agent ∈ team do
            abilityScore ← score based on agent's skills needed to perform task
            opportunityScore ← environment-dependent score based on agent, task and
              globalContext
            totalScore ← t-norm on abilityScore and opportunityScore
            if totalScore >= highestScore then
                bestAgent ← agent
                bestTask ← task
                highestScore ← totalScore
            end
        end
    end

    /* if the best score is too low, the allocation phase fails                    */
    if highestScore < minScore then
        fail
    end

    allocate bestAgent to bestTask in socialPlan
    apply effects of bestTask to globalContext
    remove bestTask from unallocatedTasks
    remove bestTask from tasksToAllocate
    refresh tasksReadyToAllocate
end

return socialPlan
```

5.4 Commitment Distribution

The goal of this phase is to build commitments based on the established social plan. For every task in the plan, ARAG sends a task execution request to the appropriate NPC via the Communication Layer. The commitment is made and saved in the belief base upon receipt of confirmation. If not, commitment distribution fails, non-cooperating NPCs are removed from the pool of available NPCs, and the system returns to team formation.

Algorithm 3: DistributeCommitments

```
Input: socialPlan - a plan with NPCs assigned to tasks
Data: potentialTeammates - the set of all NPCs considered for teamwork
Output: success or failure

/* attempt to form commitments in the same order that task allocation took place   */
foreach task, allocatedAgent ∈ allocations do
    request allocatedAgent to do task
    response ← response from allocatedAgent
    if response is confirmation then
        create commitment
    end
    else
        /* if the agent rejects the task or does not respond, it is removed from the
           considered set of teammates                                              */
        remove allocatedAgent from potentialTeammates
        return failure
    end
end

broadcast commitments
return success
```

5.5 Team Action Monitoring

Once commitments are distributed, the team executes the social plan. NPCs perform tasks while periodically reporting results to ARAG. ARAG coordinates team actions by monitoring and potentially reconfiguring the plan or dissolving the team to create a new plan. In summary, ARAG develops and proceeds with the social plan as follows:

1. If a social plan is underway, its status (in progress/success/failure) is determined based on reports from teammates.
2. Optional tasks are generated and filtered (see Section 4.1). As a standard solution, see [27], option generation is carried out periodically during plan execution to enable adaptation to the changing environment.
3. The best task is selected.
4. If the best task differs from the current one, a new social plan is created. Otherwise, the current plan is kept going and adjusted.
5. The entire process is repeated cyclically while maintaining task prioritizing to adjust to environmental changes.

Algorithm 4: ARAG Update

```
Data: currentTask
Data: currentSocialPlan
Data: commitments - commitments related to the plan
Data: potentialTeammates

options ← empty set
if currentSocialPlan ≠ null then
    /* determine the plan's status and add new task options based on report
       monitoring                                                          */
    planStatus, newOptions ← Monitor (currentSocialPlan, commitments)
    add newOptions to options
    if planStatus is success then
        broadcast success to all teammates
        clean up commitments related currentSocialPlan
        currentSocialPlan ← null
    end

    /* the currently executed task is always added as an option            */
    add currentTask to options
end

add tasks produced by the Opportunity Analyzer to options
bestOption ← option from options chosen by the task selector
if bestOption ≠ currentTask then
    /* the current task is replaced                                        */
    clean up commitments
    currentTask ← bestOption
    currentSocialPlan ← ARA(currentTask, potentialTeammates)
end
else if planStatus is failure then
    clean up commitments
    /* remove tasks that are already finished from the plan and reconfigure */
    remove successful tasks from currentSocialPlan
    currentSocialPlan ← ARA(currentTask, potentialTeammates,
      currentSocialPlan)
end
```

6 Discussion of Design Choices

Virtual gaming environments are combined with autonomous agents in an interdisciplinary way to solve NPC collaboration during gameplay. Considering the complexity of turn-based and real-time video games, the effectiveness of specific elements is crucial. This includes:

1. Communication. Only basic speech acts like bilateral communication and announcement are in use. Clearly, they lead to the two distinct kinds of beliefs: individual and shared ones. When contrasted to the whole capacity of collective beliefs, this simplification may be lifted in the future. Similarly, the assumption that ARAG has direct access to NPCs' beliefs will be relaxed, since ultimately beliefs should be accessed via communication.
2. Filtering. During reconfiguration, IRMA-style [3] filtering system is used to narrow down the stream of potential options. Contrary to traditional solutions, ARAG combines not just two, but three streams of optional tasks, including also a monitoring-related one. The user who is in charge of the filter content needs to be cautious about the granularity of these processes.

3. Selecting the winner. Although ARAG makes no explicit assumptions about the selection algorithm, Utility AI [14,22] is one of suitable techniques to select the winning task from among admissible ones. This method permits to aggregate a wide range of decision-related factors.
4. Group topology. The chosen star topology with ARAG as the central node, introduces centralized agent management features and reduces the degree of autonomy enjoyed by individual NPCs. Thanks to this, the multi-phase group commitment setting technique is simplified to a single round.

As our main focus is the impact of a game's adversarial features on teamwork, a systematic oversight of the progress of social plans is necessary. Therefore, in ARAG, we have committed to rigorous reporting on task statuses by the agents. In particular, a lack of a report may warrant an alert and trigger reconfiguration.

7 Proof of Concept

As a proof of concept, we developed Treasure Hunt, a stealth game [11] using the Unity engine [24]. The game features a player-controlled character and a team of NPCs led by a coordinator, controlled by ARAG. These NPCs work together to collect treasures scattered across the map, while considering their unique abilities and limited fields of vision. Meanwhile, the player tries hard to prevent them from acquiring the treasures.
The player and the NPCs can perform the following actions:

- `Movement`: the player and NPCs move freely across the map, limited only by physical obstacles.
- `Picking up treasures`.
- `Picking up keys`: special keys may be collected to unlock doors.
- `Pressure plates`: some doors require NPCs to stand on designated pressure plates to open them. Once an NPC leaves the plate, the door locks again.
- `Attack`: the player stuns NPCs by attacking them while staying outside their field of vision. Stunned NPCs cannot communicate or perform any actions.
- `Reviving NPCs`: an NPC revives a stunned teammate by touching it.

The NPCs play two distinct roles: *robbers* and *locksmiths*, each capable of performing different actions. Only *locksmiths* can pick up keys, while *robbers* are capable of collecting treasures, necessitating a well-formed team.

The behavior of NPC teams has been assessed in five scenarios (described in more detail in the appendix):

- Scenario 1: the player intentionally avoids the NPCs.
- Scenario 2: the player attempts to enter an NPC's field of vision, triggering a pursuit. Meanwhile, another teammate completes the original NPC's task.
- Scenario 3: an NPC is stunned by the player, prompting another NPC to pause its task, revive the ally, and resume the plan without reconfiguration.

– Scenario 4: the player stuns enough NPCs to render the current plan infeasible, leading the team to carry out an alternative plan.
– Scenario 5: following a successful attack by the player, a new team is created to accomplish the task.

Our findings demonstrate that ARAG significantly enhances the player's gaming experience. Without teamwork, the behavior of NPCs would be of much lower quality, resulting in a less satisfying gameplay. Importantly, collaboration with other NPCs is crucial for collecting treasures: in some cases NPCs could end up doing nothing if left to their own devices. Table 1 contrasts the behaviors of individual NPCs with those of ARAG-coordinated teams. Furthermore, the execution of ARAG does not noticeably impact the game's performance.

Individual action	Team action
Only treasures that are not secured in rooms can be gathered by NPCs.	NPCs cooperate to open doors and get access to guarded treasures.
If an NPC is stunned, its task will not be pursued by other NPCs.	ARAG monitors the task's status and reallocates the task to another NPC.
NPCs select tasks without taking others into account.	ARAG assigns tasks to the most appropriate NPCs.
Multiple NPCs might try to perform the same task.	ARAG makes sure that task allocations are unique.

Table 1: Comparison of individual and team action facilitated by ARAG.

Currently, the evaluation remains qualitative, necessitating further development to establish quantitative metrics and benchmarks for comparing competing solutions. Furthermore, our approach should be applied to diverse adversarial video game types in order to demonstrate its generalizability.

8 Related Work

Teamwork is an active area of research in multiagent systems for three decades already. Building on foundations laid by Wooldridge and Jennings [28], researchers have proposed various approaches to enable effective teamwork of intelligent agents. Dunin-Kęplicz and Verbrugge [5] describe a formal approach to cooperation with collective motivational attitudes as central notions. The role of these notions is highlighted in their Reconfiguration Algorithm.

Among Tambe's contributions, a notable one is the STEAM framework [23]. It employs agents equipped with an explicit model of teamwork, enabling effective communication, monitoring and replanning. Also, the MOISE framework [8] introduces a teamwork model based on roles, groups and missions.

Despite efforts to integrate MAS approaches with game engines [13,17], teamwork in video games is largely unexplored. Still, there have been many successful attempts to achieve agent coordination in a centralized fashion. In a typical solution, a single agent assigns tasks to subordinate agents that are not equipped with individual beliefs. Siemonsmeier describes a multi-agent planning in a popular tactical game, Gears Tactics [21]. Some aspects of teamwork, like action allocation has been studied in the context of real-time strategy games, however, also in a centralized fashion [20]. Despite their effectiveness, these centralized approaches are not suitable for simulating believable behaviors in environments characterized by imperfect information, as they disregard the aspect of limited senses and changing beliefs of individual agents.

Agis et al. [1] incorporated communication and coordination mechanisms into Behavior Trees to simulate a rudimentary kind of teamwork. This approach can handle simple scenarios that do not require planning or efficient task allocation.

Multi-agent systems inspired video games designers in solving problems not solely related to NPC behavior. For example, Lora used MAS related methods to design games by conceptualizing them as a system of interacting agents and defining relevant rules in predicate logic [4].

Convincing simulation of group behaviors seems to be neglected in video game literature. In fact, most research focused on action planning, whilst team formation, intra-group communication and belief management do not get enough attention [2]. This is remarkable given the amount of attention received by Park et al. [16] publication on behavior simulation. They used a Large Language Model to produce believable behaviors of NPCs.

9 Conclusions and Future Work

Our interdisciplinary approach to integrating teamwork with adversarial video games greatly simplifies the implementation of NPC collaboration. As an innovative application of MAS-related ideas it demonstrates a fresh attempt to the often-neglected area of NPCs' collaboration. Importantly, video game criteria are different from those of MAS as they aim to provide a convincing appearance of intelligent, human-like behavior rather than to find the best solution.

In this work, we focused on collaboration within a single star group topology. We plan to extend our approach to encompass less centralized group topologies and to consider multiple competing teams. Importantly, video game research is also relevant to robotics, where human-robot and robot-robot interactions face similar challenges related to embodiment and limited senses.

ARAG has been implemented and tested in a prototype stealth game, proving that our approach results in convincing behaviors of NPC groups. Currently the game is further developed to showcase more advanced teamwork scenarios. While the advantages of incorporating teamwork into video games are mostly qualitative and depend on the player's perception of NPC behaviors as well as their subjective enjoyment of playing the game, we are now developing methods to assess ARAG's quality in a more objective manner.

Acknowledgements

The authors express gratitude to Andrzej Szałas for his insightful comments, which greatly contributed to the improvement of this work.

References

1. Agis, R.A., Gottifredi, S., García, A.J.: An event-driven behavior trees extension to facilitate non-player multi-agent coordination in video games. Expert Systems with Applications **155**, 113457 (2020)
2. Barambones, J., Cano-Benito, J., Sánchez-Rivero, I., Imbert, R., Richoux, F.: Multi-agent systems on virtual games: A systematic mapping study. IEEE Transactions on Games (2022)
3. Bratman, M.E., Israel, D.J., Pollack, M.E.: Plans and resource-bounded practical reasoning. Computational intelligence **4**(3), 349–355 (1988)
4. C., L.M.L.: Game Development Based on Multi-agent Systems. Ph.D. thesis, Universitat Jaume I (2022)
5. Dunin-Kęplicz, B., Verbrugge, R.: Teamwork in Multi-Agent Systems. A Formal Approach. John Wiley & Sons, Ltd. (2010)
6. Grant, J., Kraus, S., Perlis, D.: Formal approaches to teamwork. (2005)
7. Gregory, J.: Game engine architecture. Taylor & Francis Ltd., 1 edn. (2009)
8. Hannoun, M., Boissier, O., Sichman, J.S., Sayettat, C.: Moise: An organizational model for multi-agent systems. In: Ibero-American Conference on Artificial Intelligence. pp. 156–165. Springer (2000)
9. Van der Hoek, W., Wooldridge, M.: Multi-agent systems. Foundations of Artificial Intelligence **3**, 887–928 (2008)
10. Knuth, D.E.: The art of computer programming, volume 4A: combinatorial algorithms, part 1, Algorithm R. Pearson Education India (2011)
11. Konzack, L.: Video game genres. In: Encyclopedia of Information Science and Technology, Third Edition, pp. 3070–3076. IGI Global (2015)
12. van Linder, B., van der Hoek, W., Meyer, J.C.: Formalizing abilities and opportunities of agents. Fundamenta Informaticae **34**(1-2), 53–101 (1998)
13. Mariani, S., Omicini, A., et al.: Game engines to model mas: A research roadmap. In: WOA. pp. 106–111 (2016)
14. Mark, D.: Behavioral mathematics for game AI. Course Technology Cengage Learning (2009)
15. Mirsky, R., Carlucho, I., Rahman, A., Fosong, E., Macke, W., Sridharan, M., Stone, P., Albrecht, S.V.: A survey of ad hoc teamwork research. In: Baumeister, D., Rothe, J. (eds.) Proc. EUMAS' '2022. LNCS, vol. 13442. Springer (2022)
16. Park, J.S., O'Brien, J., Cai, C.J., Morris, M.R., L., P., Bernstein, M.S.: Generative agents: Interactive simulacra of human behavior. In: Proceedings of the 36th annual acm symposium on user interface software and technology. pp. 1–22 (2023)
17. Poli, N.: Game Engines and MAS: BDI & artifacts in Unity. Ph.D. thesis, Alma Mater Studiorum Universita di Bologna Bologna, Italy (2018)
18. Politowski, C., Petrillo, F., Montandon, J.E., Valente, M.T., Guéhéneuc, Y.G.: Are game engines software frameworks? a three-perspective study. Journal of Systems and Software **171**, 110846 (2021)
19. Pynadath, D.V., Tambe, M.: Multiagent teamwork: analyzing the optimality and complexity of key theories and models. In: Proc. AAMAS'2002. ACM (2002)

20. Rogers, K.D., Skabar, A.A.: A micromanagement task allocation system for real-time strategy games. IEEE Transactions on Computational Intelligence and AI in Games **6**(1), 67–77 (2014)
21. Siemonsmeier, M.: Gearing the tactics genre: Simultaneous ai actions in gears tactics. gameaipro.com (2021)
22. Świechowski, M., Lewiński, D., Tyl, R.: Combining Utility AI and MCTS towards creating intelligent agents in video games, with the use case of tactical troops: Anthracite shift. In: 2021 IEEE Symposium Series on Computational Intelligence (SSCI). pp. 1–8. IEEE (2021)
23. Tambe, M.: Towards flexible teamwork. Journal of artificial intelligence research **7**, 83–124 (1997)
24. Unity Technologies: Unity (2023), `https://unity.com/`, game development platform
25. Warpefelt, H.: The Non-Player Character: Exploring the believability of NPC presentation and behavior. Ph.D. thesis, Department of Computer and Systems Sciences, Stockholm University (2016)
26. Wooldridge, M.: Reasoning about rational agents. MIT press (2003)
27. Wooldridge, M.: An introduction to multiagent systems. John wiley & sons (2009)
28. Wooldridge, M., R., J.N.: The cooperative problem solving process. Journal of Logic and Computation (1999)