Noname manuscript No.
(will be inserted by the editor)

Minimap: An Interactive Dynamic Decision Making Game for Search and Rescue Missions

Thuy Ngoc Nguyen · Cleotilde Gonzalez

Received: date / Accepted: date

Abstract Many aspects of humans' dynamic decision-making (DDM) behaviors have been studied with computer-simulated games called microworlds. However, most microworlds only emphasize specific elements of DDM and are inflexible in generating a variety of environments and experimental designs. Moreover, despite the ubiquity of gridworld games for Artificial Intelligence (AI) research, only some tools exist to aid in the development of browser-based gridworld environments for studying the dynamics of human decision-making behavior. To address these issues, we introduce *Minimap*, a dynamic interactive game to examine DDM in search and rescue missions, which incorporates all the essential characteristics of DDM and offers a wide range of flexibility regarding experimental setups and the creation of experimental scenarios. Minimap specifically allows customization of dynamics, complexity, opaqueness, and dynamic complexity when designing a DDM task. Minimap also enables researchers to visualize and replay recorded human trajectories for the analysis of human behavior. To demonstrate the utility of Minimap, we present a behavioral experiment that examines the impact of different degrees of structural complexity coupled with the opaqueness of the environment on human decision-making performance under time constraints. We discuss the potential applications of Minimap in improving productivity and transparent replications of human behavior and human-AI teaming research. We made Minimap an open-source tool, freely available at `https://github.com/DDM-Lab/MinimapInteractiveDDMGame`.

Keywords Dynamic decision making · Online experiments · Microworlds · Search and rescue tasks · Open source · Open science

1 Introduction

The world is full of dynamic decision-making (DDM) challenges, typically characterized by the need to make sequential and interdependent decisions under uncertainty. Subsequently, the environment changes

Thuy Ngoc Nguyen · Cleotilde Gonzalez
Carnegie Mellon University, 5000 Forbes Ave., Pittsburgh, 15213 PA, USA
E-mail: coty@cmu.edu

as a consequence of these decisions and also evolves independently [7,9]. More concretely, an environment is a DDM task if it possesses four characteristics, namely *dynamics* (the dependence of the environment's state on its state at an earlier time), *complexity* (the number of components in the environment, the number of relationships among them and the nature of these relationships), *opaqueness* (the ambiguity, invisibility of some aspects in the environment) and *dynamic complexity* (the inclusion of feedback delays and interrelationships of decisions over time) [25].

Many naturalistic decision-making settings are DDM tasks [22,36,18]. One prominent example we consider here is the domain of urban search and rescue. In such environments, one inevitably must cope with multiple possibilities and alternatives and weigh multiple attributes (*complexity*) accompanied by a great deal of uncertainty about the situation (*opaqueness*). The situation continuously changes (e.g., victims perish) independently of the human actions (*dynamic*). Furthermore, long feedback delays, which often prevail, generate *dynamic complexity*, as it is challenging to associate outcomes with specific actions over time [27]. In addition, time constraints and the interrelationships of decisions over time are inherent in the search and rescue of victims in an environment that evolves in real time.

Making decisions in such DDM environments presents a considerable challenge for humans. Indeed, previous findings suggest that humans generally have difficulty processing feedback delays [7,6] and associating their actions with the results they observe over time [28]. Past research has also shown that people's performance deteriorates in situations involving time constraints [23,35] and workload [29]. Studying how people make decisions in naturalistic situations, such as urban search and rescue, is highly complex, time-consuming, and expensive [36]. Therefore, a large body of research in the field of DDM has employed computer-interactive simulations, often referred to as "microworlds" [9,30,26,46], to improve our understanding of the process by which humans learn to make decisions in these situations of DDM.

Most existent microworlds expressly represent a subset of DDM characteristics, such as uncertainty or complexity. They have yet to support the investigation of all essential features of most real-life decision environments, including real-time decision processes, time constraints, and interdependence between decisions over time [30]. Moreover, a large body of microworlds caters to specific scenarios, with little consideration of offering the flexibility to generate various decision situations. The potential of browser-based microworld for online data collection has also been limited. Finally, 2D gridworld environments are the prevailing paradigm in the field of AI for studying diverse decision making problems, such as navigation and exploration [51]. 2D gridworlds are simulation environments represented as a 2D grid wherein each cell represents a state. Players can move from one state to another by taking specific actions, such as moving up, down, left, or right. However, 2D gridworlds are mostly used as simulation tools [3,11], and web-based gridworlds that can be used in experiments for the research of human DDM have yet to be fully explored.

To address these challenges, we introduce *Minimap*, a dynamic interactive web-based application for studying decision-making behavior in search and rescue missions. Unlike most extant microworlds, Minimap features *all the essential characteristics* of a DDM task. Specifically, it presents a decision maker with a sequential decision problem in which the environment evolves autonomously and in response to their decision (*dynamics*). The decision maker has to make decisions within time constraints in a complex structure, and multiple rescue mechanism settings (*complexity*) coupled with uncertainty about the environment (*opaqueness*) and long feedback delays with rewards happen over long time horizons (*dynamic complexity*). Unlike existing simulated search and rescue tasks in the literature, Minimap is an open-source project for *2D gridworld* based research. Its design exploits the full capabilities of *modern browsers for interactivity* for online data collection from humans and for behavioral analysis. Importantly, Minimap provides *scenario customization*, in which the current task set can appropriately adapt to the experimenter's needs. That is, it is fully customizable and offers flexibility to experimenters in defining experimental setups, manipulating experiment scenarios, varying aspects of DDM situations, and collecting data.

2 Related work

Computerized microworlds. To date, many microworlds have been developed in an effort to investigate human behavior in a large variety of tasks ranging from simple gambling games [14] to complex control task settings, including dynamic stock and flow systems [26], military command and control [41], air traffic control [39,20,10], resource allocation [46,24], and supply chain management [54,13]. However, most developed microworlds focus exclusively on some aspects of DDM tasks by providing specific environments for specific problems. Furthermore, only a handful of studies offer flexibility in generating various decision scenarios and experimental designs [46,26]. FireChief [46] was one of the computer-simulated microworlds developed as a generic framework for use in different contexts requiring a complex task situation. Nevertheless, it has yet to support the customization of all the features of DDM tasks. The development of FireChief was about two decades ago without having the possibility of tapping into the advances of web-based technologies. Recently, Minecraft video games have been employed as a source of computer-simulated microworlds to study human-AI teaming in a search and rescue task environment [12, 38,38]. Relatedly, the Malmo project is a platform for AI experimentation and research built on top of Minecraft [33]. Malmo exposes human participants and agents to complex 3D environments with varied gameplay. Minerl [31] is another project designed to support AI research. Yet, as its name suggests, MineRL is primarily built to improve the development of deep reinforcement learning agents rather than to study human behaviors. Arguably, the structure of these 3D environments and video games-based simulations are hard to readily adapt for the precise experimental study of complex decision-making behavior since the functionality to modify experiment parameters is often unavailable. Aside from that, deploying experiments in 3D environments with Malmo or Minerl projects require high resources and tech-savvy skills, adding another complication to running behavioral research. Furthermore, video games are usually so sophisticated that many hours of play are needed to achieve basic competency in the task, making them unsuitable for addressing diverse research questions in various large-sample populations [46].

Online experiments for behavioral research. Online experiments are popular ways to study human decision-making behavior, as they overcome the hurdles of setting a computer-simulated microworld in a laboratory environment with rapid and low-cost data collection. With the advances in web technology, creating many different kinds of computer-based behavioral experiments in a web browser is now feasible. Thus, online experiments have become a popular way to collect data in the social and behavioral sciences. A variety of web-based libraries and platforms are available to facilitate the process of building, running, and collecting experimental data, accommodating various needs such as jsPsych [15], PsychoPy [50,49], lab.js [32], or PsyToolkit [55]. Even though the purpose of these tools is to provide a set of templates for common research scenarios, they have yet to explicitly support 2D gridworld tasks that are ubiquitous in the field of AI.

Gridworld games. A 2D gridworld environment is constructed based on a two-dimensional grid in which each position (tile or cell) is uniquely identified by a coordinate tuple (x, y). The (x, y) coordinates are nonnegative integers. Each tile is typically associated with an entity, such as a player, a door, a wall, or an obstacle. The 2D environment can also contain rewards and other features, which are beneficial for examining agents' behavior, and hence numerous 2D gridworld environments for AI research have been developed to study diverse problems such as navigation, imperfect information, exploration, and so on [11,3,51]. However, these widely used task environments, such as those in Atari Learning Environment Benchmark [4] appear to be tailored for a specific single environment without customization of the task, making it difficult to change values, appearance, or integrated components in tasks. Consequently, other recent work has focused on providing highly customizable gridworld settings, including MiniGrid [11], DMLab2D [3], and GriddlyJS [2,1]. Nevertheless, these tools are simulation environments primarily designed to provide a flexible and scalable framework that enables researchers to create and test the robustness and generalization of reinforcement learning (RL) agents in varied simulated environments rather than catering to human behavioral research.

In this work, we present Minimap, an open-source, interactive platform for studying human behavior in DDM tasks that incorporates all essential characteristics of DDM, distinguishing it from existing computerized microworld and RL-based research simulation tools. It is a 2D gridworld game and offers flexibility regarding experimental setups and creating task scenarios. Minimap also streamlines the development of web-based 2D gridworld environments for DDM research by allowing task customization of graphical interfaces. Table 1 shows the comparison between our Minimap and other similar tools.

Table 1: Comparison between Minimap and other similar tools. Minimap provides a broader range of support regarding task customization of four DDM characteristics and the development of gridworld environments, human behavior research, and browser-based interactivity.

		Minimap	FireChief [46]	MiniGrid [11]	DMLab2D [3]	GriddlyJS [2]
Configurable	Dynamics	✓	✓			
	Opaqueness	✓		✓	✓	✓
	Complexity	✓	✓	✓	✓	✓
	Dynamic Complexity	✓	✓	✓	✓	✓
Gridworld environment		✓		✓	✓	✓
Human behavior research support		✓	✓		✓	
Browser-based interactivity		✓				✓

In the next section, we present Minimap and describe all its highlights to study the characteristics of DDM in search and rescue missions. We demonstrate the use of Minimap in an online experiment where we investigate the effect of structural complexity in association with opaqueness (i.e., uncertainty about the environment) on human decision behavior. We also discuss how Minimap can accommodate various behavioral research questions, from human behavior studies to human-AI interaction.

3 Minimap: A Game to Study Search and Rescue Missions

Minimap is implemented to take into account all the characteristics of a DDM task with the purpose of studying human learning and decision-making in a generic search and choice task. Search and rescue task is selected as the default scenario because it is judged to be illustrative of a realistic DDM task, inherently interesting, and readily understood by human subjects. Nevertheless, the development of Minimap aims to be fully customizable and extensible, enabling experimenters to systematically manipulate or vary different aspects of complex and dynamic situations.

3.1 A Search and Rescue Scenario: Generic Instructions

The generic version of the search and rescue task in Minimap is designed to simulate typical task constraints that a rescuer might encounter[1]. Presumably, the instructions will vary according to the specific parameter values and features that have been selected.

The mission aims to obtain as many points as possible by finding and rescuing victims from inside a building that collapses within a 5-minute time limit. The game's layout is illustrated in Fig. 1. That is, the rescuer must navigate through an environment and avoid obstacles to locate and save victims. Throughout the mission, the rescuer is informed about the time left in an episode (i.e., five-minute mission) and the accumulated points (i.e., "Points") by rescuing the victims during each episode.

The environment of the Minimap task is a 93 x 50 grid, representing one floor of this building. In each position in the grid or tile, the rescuer may encounter an obstacle, a door, a wall, a victim, or an

[1] Demo of the game: `http://janus.hss.cmu.edu:5701/demo/`

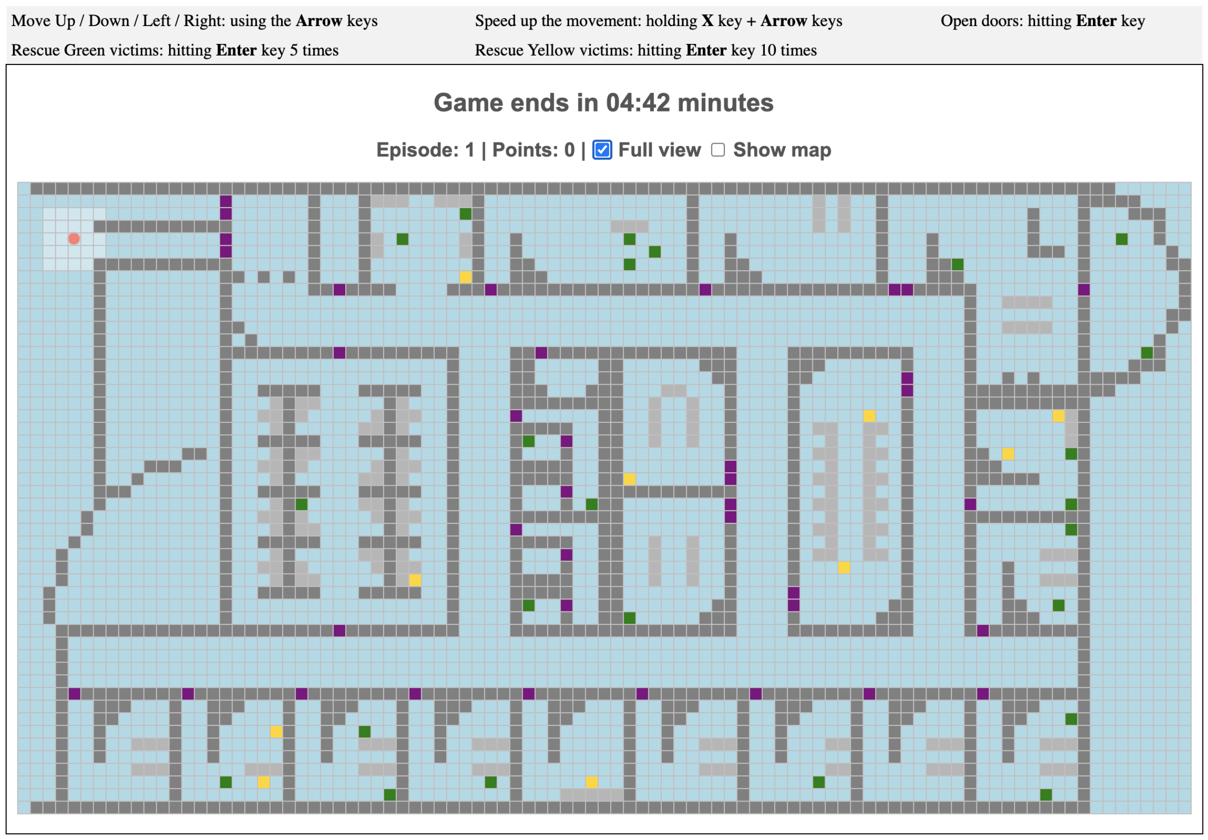

Move Up / Down / Left / Right: using the **Arrow** keys
Speed up the movement: holding **X** key + **Arrow** keys
Open doors: hitting **Enter** key
Rescue Green victims: hitting **Enter** key 5 times
Rescue Yellow victims: hitting **Enter** key 10 times
Game ends in 04:42 minutes
Episode: 1 | Points: 0 | ☑ Full view ☐ Show map

Fig. 1: The graphical user interface of Minimap for a search and rescue task with the full view mode. The red dot represents the current position of a player in the game. Green and yellow squares represent unsaved victims, grey squares represent walls, light grey squares represent obstacles, and purple squares are designated for doors. The wall and obstacle are not passable, while the empty blue squares are areas in which the player can traverse.

empty tile. There are 34 victims scattered throughout the building, of which ten are severely injured and need a more urgent evaluation, while the other 24 are not considered critically injured. This feature is simulated as critical and regular victims, yielding high and low rewards. Concretely, regular victims are worth ten points each, are represented by green tiles, take a shorter time (five button presses) to save, and are available to be rescued throughout the entire mission. By contrast, critical victims are worth thirty points each, are represented by yellow tiles, require a longer time (ten button presses) to be rescued, and they disappear from the building (or die) one minute before the mission is over.

In addition, obstacles and walls are placed in the path of the victims, forcing the rescuer to look for alternative routes. Moreover, rooms have closed doors requiring the rescuer to open them to enter. To save victims or open doors, the rescuer must be in a tile adjacent to the victim or the door. See Figure A2 in the Appendix A for detailed game keyboard controls.

By default, the rescuer is able to see objects within a 5x5 square area around them called the “vision area” or “field of view” (FoV). Depending on different degrees of environmental visibility, they may also be able to observe the walls in the building.

3.2 Characteristics of a DDM Task in Minimap

We now explain how Minimap features all four essential characteristics of DDM environments [30,25], which is one of its main highlights. For a detailed walkthrough of how to customize these DDM features in Minimap, see Appendix B.

3.2.1 Dynamics

Minimap embodies temporal dependencies among system states. The environment is dynamic with respect to players' activity. At each time t, the player takes action (e.g., open a door or save a victim), and then the environment transitions to a new state. In other words, the situation with subsequent activity taken in the environment (at time t) is partly dictated by previous activity (at time $t-1$). This feature is one of the fundamental characteristics of dynamic systems [17].

In the generic search and rescue mission, Minimap offers different dynamics wherein the environment evolves not only in response to the player's decision but also autonomously. That is, victims die and disappear from the environment in the absence of any decision input from human players. Minimap also enables the creation of sudden perturbations, which attributes to the dynamic degree of the task. The sudden changes supported in the Minimap are structural in the environments in the middle of the game, including adding obstacles, changes in the number of victims, as well as a player's and victims' locations. These perturbations again take place regardless of any decision input from players.

Minimap's dynamics degree can be easily varied by changing the environment continuously or at specific intervals. For instance, the experimenter can study humans' responses to the dynamics by manipulating the time victims die to make the disappearance (death) of victims occur quickly (e.g., every second) or slowly (e.g., every two minutes). Minimap also has the ability to implement multiple classes of perturbations and control the time when the perturbation happens during gameplay.

3.2.2 Complexity

Complexity is a relative term and can arise from various sources [53,27]. In cognitive and AI research, complexity has mainly been studied as an issue wherein a task possesses a large number of interrelated elements to be processed within a unit of time [40,47]. Complexity can also arise from the presence of multiple alternative decisions with multiple attributes [53] or the tension between the costs and benefits of the decision [44].

Minimap offers a wide range of flexibility in manipulating this kind of complexity. In particular, a different number of components can be added to the environment, as well as their placement, called *structural* or *environmental complexity*. In addition, time constraints to rescue victims are also variables in Minimap. Taken together, one can quickly increase the complexity by adding more obstacles and shortening the time to search for victims.

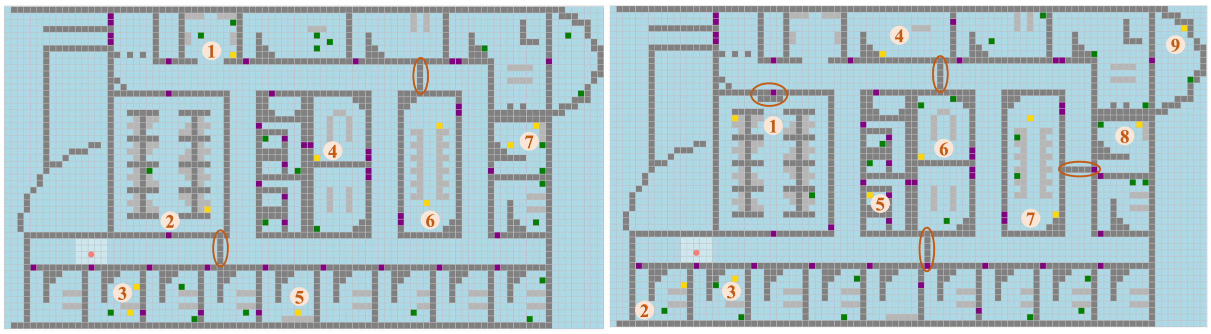


Fig. 2: Illustration of different levels of structural complexity with Minimap. The simple setting (on the left) consists of two extra blocks (encircled in red) and critical victims in seven rooms. On the contrary, the complex environment (on the right) has four additional blocks and critical victims scattered over nine rooms.

A scheme of rewards and costs is readily configurable in Minimap. Take the generic version of the search and rescue scenario as an example; the yellow and green tiles represent critical and regular victims who give high and low rewards/costs, respectively. Saving critical victims is worth more points than saving regular victims, as it requires a longer time (more effort) to be rescued. With that, one can create any mission and scenario that controls not only structural complexity but also decision complexity determined by a tension between benefit and cost in gridworld task [44].

As an example, Fig. 2 illustrates a search and rescue game with varying degrees of structural complexity determined by the number of obstacles and the difficulty of reaching critical victims. First, additional blockages (encircled in red) are placed in the middle of a path to restrict the individual from taking the blocked route, forcing them to look for longer ways to bypass the blockage. Consequently, the structural complexity of the task increases when there are more blockages. Furthermore, different levels of structural complexity differ in the placement of critical (yellow) victims. Critical victims in the complex environment are scattered farther away from the player's starting point, making them more challenging to find. Concretely, the yellow victims are spread over nine rooms in the complex environment, while in the simple setting, they are located in seven rooms. Therefore, it requires the rescuer's ability to search through the building.

3.2.3 Opaqueness

A task becomes challenging in the presence of uncertainty about the environment. Thus, the essential element of DDM tasks is opaqueness which arises when some aspects of the environment are invisible to a decision-maker. The opaqueness or ambiguity of the environment is typically represented by the information available about the state of the world [56,43].

Minimap fully incorporates this feature as it conceals the information on the number of components in the environment from the players. Consequently, they must explore the environment by interacting with it and gradually learn about these unknown parts of the environment. Importantly, Minimap allows customization of the visibility variable at different levels of granularity. Concretely, the degree of opaqueness is configurable with three levels, namely the field of view (*fov*), map view (*map*), and full view (*full*). With the field of view, the individual can only observe part of the environment within their restricted vision area. In contrast, with the map view, they can see the overview structure of the map in addition to their vision area. Conversely, the full view gives the player a complete view of the environment, including where the victims are. Fig. 3 illustrates the two levels of opaqueness: Fov and map view.

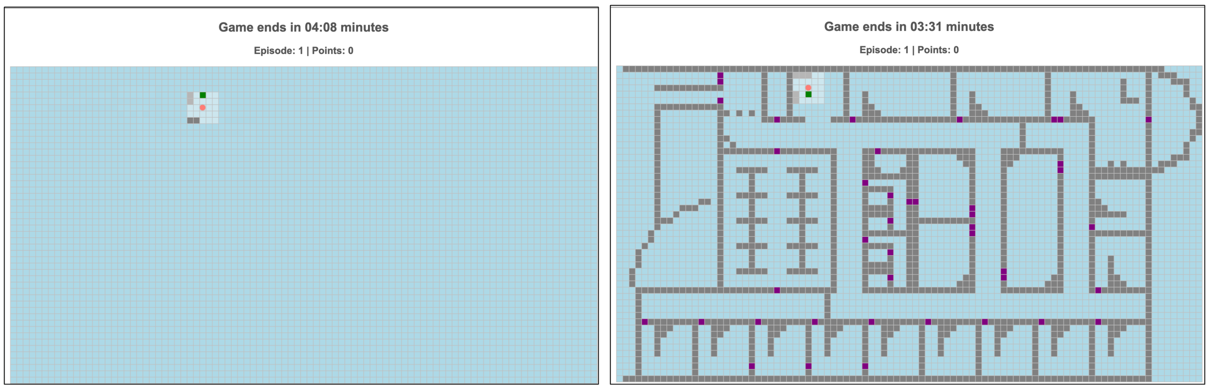


Fig. 3: Illustration of the Minimap with two degrees of opaqueness from left to right: FoV and map view.

3.2.4 Dynamic Complexity

Dynamic complexity varies in their inclusion of delayed feedback [16] since the increase in feedback delays between decision actions and the corresponding decision outcomes negatively affect human performance in various dynamic tasks [8,25].

Feedback delays are inherent in Minimap, a sequential decision-making task. That is, the user has to make a sequence of decisions without feedback, knowing the result only at the end of the sequence. For example, in the search and rescue scenario, a decision maker must navigate the space by deciding which actions to take while unaware of the locations of victims and obstacles. Also, the large dimension of the environment space makes for long feedback delays, and rewards happen over long time horizons until the end of the game duration.

Similar to the other characteristics, the feature of feedback delays is modifiable in Minimap. Technically, the delay in the individual's action is defined by a variable configurable by the experimenters. The variable adds a constant number of time intervals k to the effect of an action. In other words, the action only takes effect after the time interval k has passed.

3.3 Overall Structure of Minimap

We now describe the Minimap technical architecture to give a basic idea of how it has been developed and highlight additional features of Minimap.

3.3.1 Minimap's Architecture

As illustrated in Fig. 4, Minimap follows a client-server architecture consisting of two parts: client browsers and a Minimap web server. On the server side, Minimap is built on the FastAPI[2] web framework in Python. The Minimap client-side relies on open Web technologies (i.e., HTML5, CSS, JavaScript). It also leverages third-party libraries such as jQuery[3], p5.js[4] for graphical design and socket-io client [5] to facilitate real-time interaction with a web browser or other client applications.

Minimap adopts the RESTful API web service for communication between client and server. REST APIs are well known for scalability and flexibility, such that an application using REST APIs can be scaled quickly due to the separation between the client and the server. Minimap also uses WebSockets as an additional protocol on top of HTTP to keep a long-running connection open so that a stream of messages can be sent over a long period without having to set up a new connection for each request.

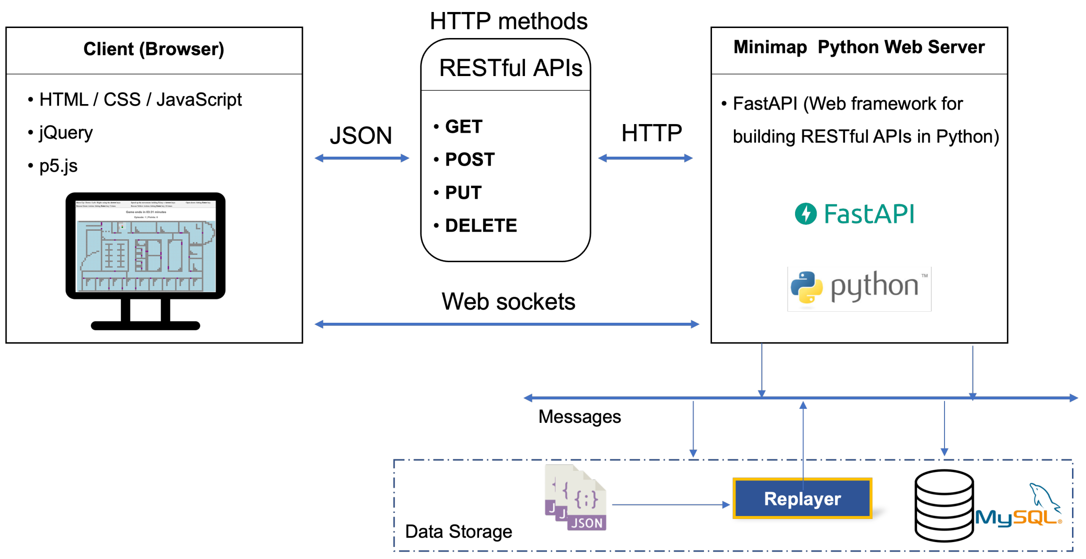


Fig. 4: Schematic representation of the Minimap architecture with two main components: Web server and client. The client sends a request in the form of JSON data, and the server sends a response through RESTful APIs methods.

Finally, Minimap uses the MySQL database management system to store data collected from human subjects. Trajectories of humans playing during the game are recorded once human participants execute

[2] a modern, high-performance, web framework for building APIs with Python 3.6+ (`https://fastapi.tiangolo.com/`).
[3] JavaScript library designed to simplify the client-side manipulation of an HTML page (`https://jquery.com/`).
[4] JavaScript library for drawing functionality using the HTML5 canvas element (`https://p5js.org/`).
[5] Socket-io client library (`https://socket.io/`)

```
{
    "x": 26,
    "y": 2,
    "timestamp": 1667933847.90843,
    "mission_time": "00:48",
    "event": "triage green in-progress",
    "score": {"green": 0, "yellow": 0}
}
```

Listing 1: Example of an object of the `JSON` array stored in a data file

an event occurrence. Otherwise, human trajectories are automatically saved in the database every 30 seconds, regardless of any input from them. In addition, Minimap automatically records human trajectories every second and saves them as a `JSON` file. These recorded trajectories can be replayed inside Minimap.

Overall, Minimap's design aims at being cross-browser and platform agnostic. It can facilitate online participation because users can join an experiment directly in the browser window without installing additional software. This feature distinguishes Minimap from other search and rescue tasks simulated in the Minecraft video game [12,38,21].

3.3.2 Human Trajectory Replay

Visualizing humans' actions is crucial for DDM research that studies human decision behavior. By default, Minimap automatically saves players' data for each episode in a `JSON` format. Each object of the `JSON` array stores information about the player's action in a specific `timestamp` associated with `mission time`. The information includes the position (x, y) coordinate of the player, the `event` that the player interacts with the environment, and the `score` at the timestamp.

Minimap provides a point-and-click interface that allows experimenters to select a `JSON` data file that will be useful to replay (see Fig. 5). During the replay, the experimenter can pause and resume replay execution as desired. As such, the replay facility allows experimenters to identify strategies, strengths, weaknesses, and unexpected behaviors. It also provides insight into any problems with data collection.

3.3.3 Task Scenario Customization

Flexibility and generality are integral in microworld-generating programs. Flexibility facilitates the development of various microworlds to address a wide range of research questions in many scenarios. At the same time, this generality enables researchers to validate the extent to which the obtained findings are still applicable to other environments. Therefore, Minimap supports a quick design of new 2D gridworld tasks or variations of an environment without the need for specialized technical knowledge.

Minimap, by default, features the environment as illustrated in Fig. 1. However, the environment design is fully customizable. That is, one can entirely customize the Minimap layouts, including the

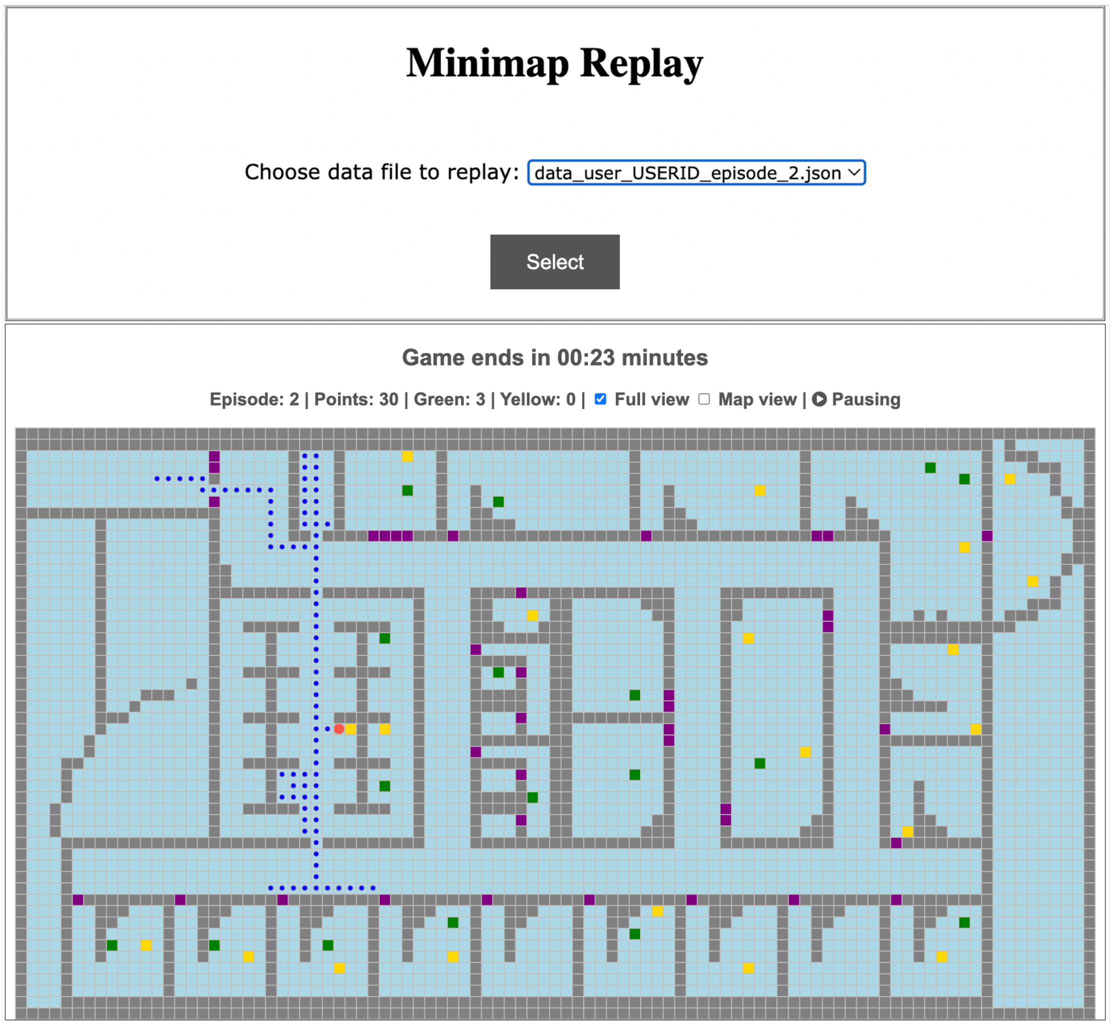
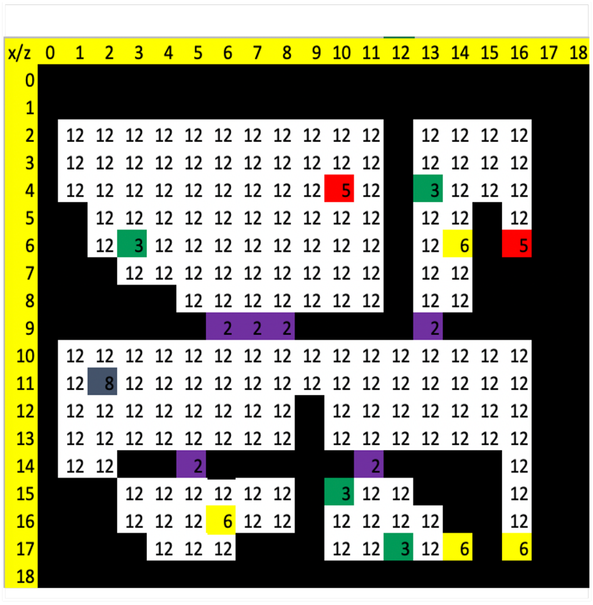

Minimap Replay
Choose data file to replay: data_user_USERID_episode_2.json
Select
Game ends in 00:23 minutes
Episode: 2 | Points: 30 | Green: 3 | Yellow: 0 | ☑ Full view ☐ Map view | Pausing

Fig. 5: Screeonshot of the replay function. It also allows different viewing modes ("Full view" or "Map view"). One can pause and resume replay execution as desired using the toggle button "Playing/Pausing".

environment size, objects' location, and the number of entities presented on the map (e.g., victims, walls, obstacles, and doors) to create diverse environments of task scenarios. Fig. 6 illustrates a new task design with Minimap. As discussed earlier, generating a new task scenario in Minimap can be achieved without requiring specialized programming knowledge. Remarkably, one can modify a base map designed in a `Excel` format provided within the Minimap project by using the appropriate codebook (see Fig. 6a) and then save it in the `CSV` format. Minimap will load the newly designed map in `CSV` to display it in the game (see Fig. 6b). For a detailed and concrete example of how to customize a task and its interface layout, see the Appendix B.7.

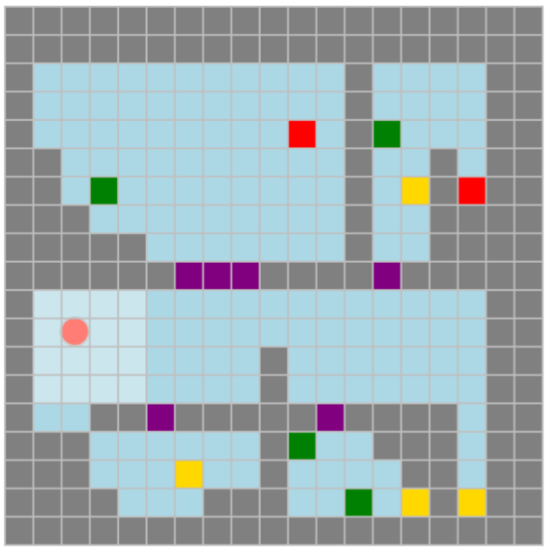
4 Proof of Concept: Experiment on Complexity and Opaqueness with Minimap

We demonstrate the utility of Minimap by conducting a behavioral experiment that varies in structural complexity and opaqueness under time constraints to investigate its effects on human decision-making behavior. More concretely, the structural complexity is determined by the number of additional obstacles



x/z	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0																			
1																			
2		12	12	12	12	12	12	12	12	12	12	12		12	12	12	12		
3		12	12	12	12	12	12	12	12	12	12	12		12	12	12	12		
4		12	12	12	12	12	12	12	12	12	5	12		3	12	12	12		
5			12	12	12	12	12	12	12	12	12	12		12	12		12		
6			12	3	12	12	12	12	12	12	12	12		12	6		5		
7				12	12	12	12	12	12	12	12	12		12	12				
8						12	12	12	12	12	12	12		12	12				
9							2	2	2					2					
10		12	12	12	12	12	12	12	12	12	12	12	12	12	12	12	12		
11		12	8	12	12	12	12	12	12	12	12	12	12	12	12	12	12		
12		12	12	12	12	12	12	12	12		12	12	12	12	12	12	12		
13		12	12	12	12	12	12	12	12		12	12	12	12	12	12	12		
14		12	12			2						2					12		
15				12	12	12	12	12	12		3	12	12				12		
16				12	12	12	6	12	12		12	12	12	12			12		
17					12	12	12				12	12	3	12	6		6		
18																			

(a) Map design in an `Excel` file



(b) Map design rendered in Minimap

Fig. 6: Example of a new gridworld task generated by Minimap

Table 2: Summary of the independent variables

Variables	Description
Complexity	Simple: two additional obstacles; critical victims are spread in seven rooms Complex: four additional obstacles; critical victims are spread to nine rooms
Opaqueness	FoV: 5×5 restricted vision area Map: 5x5 vision area in addition to the view of the map's structure

and the distance to critical victims from the player's starting position, i.e., the farther away (spread out) from critical victims to the player's starting points, the more complex the setting is. The field of vision area dictates the opaqueness the player can observe. We considered two levels of opaqueness called Field of View (FoV) and Map view of the building (Map), and two levels of structural complexity, namely *Simple* and *Complex*. Table 2 describes the difference between each level of complexity and opaqueness considered in the experiment. By following 2×2 factorial design, we created four experimental settings that result from all possible combinations between the two levels of opaqueness and the two degrees of structural complexity. Detailed instruction on replicating these four conditions is described in Appendix B.6.

Participants. A total of 195 participants completed the study and were roughly evenly distributed under experimental conditions. All participants were recruited from Amazon Mechanical Turk for 30-45 minutes in a study pre-registered with the Open Science Framework[6].

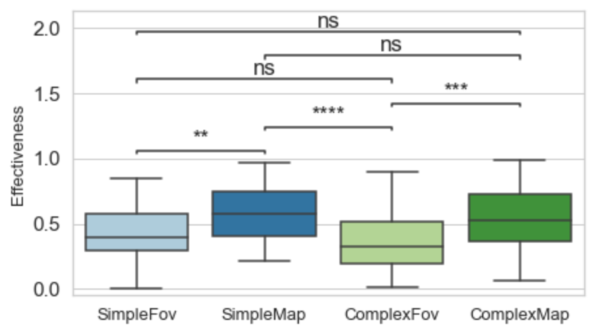
Procedure. Each participant was randomly assigned to one of the four conditions of the mission. After giving their informed consent and receiving task instructions (see Figure A1 in the Appendix A), par-

[6] `https://osf.io/5gmsc/?view_only=b7b13bcae1da448e8c3a5d58ad976e34`. According to the preregistration, we manipulated another level of structural complexity, adding five extra blocks and critical victims were distributed in ten rooms. We did not observe any results standing out in this condition compared to the complex setting; thus, we do not report them in this work.

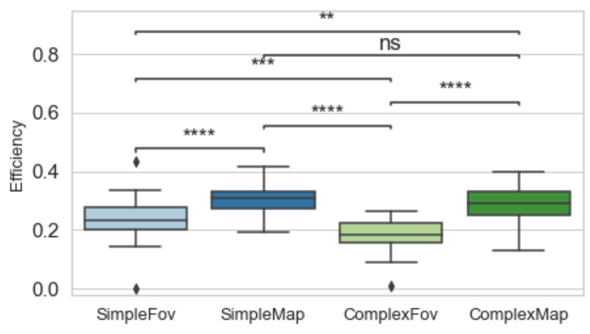
ticipants performed the study in two phases: (1) practice session and (2) main task. Subsequently, they completed a post-survey and demographic questionnaire.

In the practice session, all the participants were required to complete one-minute training to familiarize themselves with the controls. Then, they started the main task in which they were asked to complete eight five-minute Minimap episodes (that is, eight opportunities to perform the mission, and each mission episode lasted five minutes). The experimental conditions determined by the visibility and complexity of the environment remained constant for each participant throughout the study. The participants started at the exact location across the episodes.

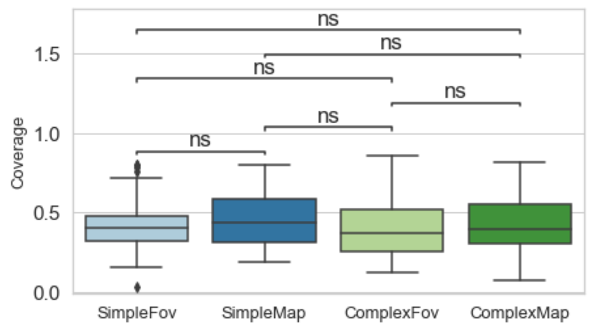
Results. We measured human performance in terms of (i) *effectiveness* (the average proportion of victims rescued in each episode); (ii) *efficiency* (the ratio between the points earned and the number of steps taken to rescue the last victims); (iii) *coverage* (the ratio of the number of unique locations to the total accessible locations on a given map); and (iv) redundancy (the proportion of the number of revisited locations to the number of uniquely visited locations).



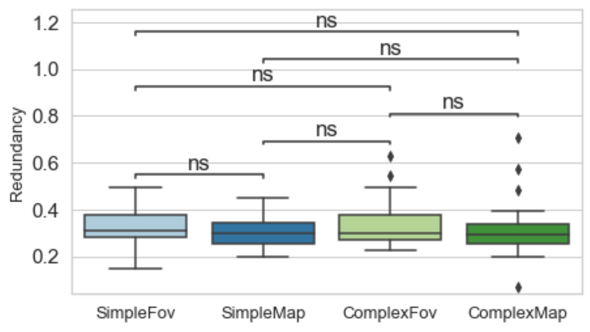
(a) Effectiveness



(b) Efficiency



(c) Coverage



(d) Redundancy

Fig. 7: Results in effectiveness, efficiency, coverage, and redundancy for each of the four experimental conditions. We applied a non-parametric Mann-Whitney-Wilcoxon test with Bonferroni correction for independent samples for statistical significance. The asterisk(*) indicates that the difference is statistically significant, while “ns” denotes the difference is not significant. The more asterisks, the lower the p-value. The small diamonds represent outliers of data points in the boxplot.

Fig. 7 shows the average effectiveness, efficiency, coverage, and redundancy of human participants for each experimental condition of the task. We observe that, regardless of complexity, the effectiveness and efficiency of participants provided with more information about the environment (map view) are significantly higher than those restricted to limited FoV. Interestingly, we did not find a significant effect of complexity on human effectiveness and efficiency in performing the task, which is counter-intuitive to the expected linear relation between environmental complexity and human performance. This result suggests that the design of task complexity that relies on structural characteristics does not necessarily lead to a more complex task that can impair human decision learning. Likewise, we did not find significant main effects of opaqueness nor redundancy on the coverage and redundancy. That is, the increase in ambiguity and structural complexity of the environment did not affect people's ability to explore the environment and their tendency to make redundant movements.

5 Discussion and Conclusions

In this paper, we introduced Minimap – an interactive browser-based microworld developed to help researchers study many aspects of human behavior in dynamic decision-making (DDM) problems and improve our understanding of how people make decisions in such environments. The primary motivation behind its development was to provide behavioral researchers with a tool to systematically manipulate or vary the different characteristics of a DDM task so that they can address specific research questions in various research contexts. Minimap enables experimenters to simulate many real-world dynamic situations featured with navigation and exploration problems. Minimap itself integrates all four essential characteristics of a DDM task: *dynamics*, *complexity*, *opaqueness*, and *dynamic complexity*. Such characteristics are fully adjustable for DDM research needs within Minimap. For example, one can easily vary the level of *opaqueness* using different visibility configurations offered in Minimap. Alternatively, the length of feedback delays (*dynamic complexity*) is also adjustable by altering the periods that an action takes effect (e.g., an action of rescuing a victim can take effect immediately or after a specific interval).

Furthermore, Minimap's development gives experimenters control over creating various scenarios and different structures of a task. For instance, Minimap, by default, is contextualized in an urban search and rescue mission, but in general, one can turn it into any mission scenario featuring navigation and exploration problems. The graphical design of the task is highly customizable without the need for specialized technical knowledge. The extent of modifications can range depending on the analogies between a new scenario and the search and rescue mission.

Crucially, unlike most existing microworlds, Minimap provides a programming framework to streamline the development of 2D gridworld games for online human behavior research. Its design exploits the full capabilities of modern browsers for interactivity, furnishing flexibility to experimenters in the definition of experimental setups (i.e., study human interactions at different time intervals) and online data collection with diverse subject populations. The Minimap source code is open and accessible for free at `https://github.com/DDM-Lab/MinimapInteractiveDDMGame`.

As a proof-of-concept, we used Minimap to examine the effects of structural complexity coupled with the ambiguity of the environment on human decision-making performance. The study's context is a simulated search and rescue mission, motivated by the fact that people in such a disaster environment often have to deal with a complex topological structure of the setting, accompanied by uncertain conditions when performing rescue operations. The results demonstrated that humans performed more effectively and efficiently when provided with the Map view than with FoV. However, structural complexity does not have a significant effect on human performance. Furthermore, people's ability to explore the environment (coverage) and their rate of making redundant movements were not affected even when the ambiguity or structural complexity of the environment increased. These results suggest the challenge of designing a complex, dynamic task at the human level. Hence, the features of Minimap benefit the generation of different DDM task scenarios at varying levels.

5.1 Implications

The experiment results can inform the design of the complexity of tasks in decisions under uncertainty. Concretely, we have learned from this work that complex structural characteristics per se do not adequately determine the complexity of a task in practice. Other elements, such as high interconnectivity, dynamically changing over time, or tension between the cost and benefits of each alternative (i.e., the complexity of decision [44]) probably play a role in dictating the task complexity. These results, thereby, highlight the utility of Minimap for designing and manipulating the DDM features at custom levels.

In addition to the ability to solve specific tasks autonomously, the ultimate objective of building autonomous systems should be to develop interactive assistant systems where humans and machines work together, taking advantage of their respective strengths [19,42]. Studies on human-autonomy teaming (HAT), in which a team of humans and intelligent, autonomous agents must work interdependently to achieve a common goal, have received ever-increasing attention in AI research [48,52]. Understanding human behavior as insights from human subjects is integral in informing and providing suggestions for autonomous system designs [37,5]. As such, Minimap facilitates the development of advisory support systems that attempt to boost human performance in many fields in which they must cope with structural complexity associated with environmental opaqueness.

Unlike existing tasks used for HAT research which focus primarily on command and control and resource management tasks, such as NeoCITIES [34], Minimap applications can be advantageous to study HAT in DDM tasks featured by navigation and space exploration problems. Furthermore, the existing tools provide no additional functionality for customization. Minimap, by contrast, allows task scenario customization. This feature facilitates the creation of environments and enables the investigation of HATs in various settings involving collaboration, attacking, and defending requirements and navigation.

Another advantage of Minmap over the extant HAT tasks is to provide an online platform for running experiments.

Indeed, recent work has utilized Minimap to capture the team's effort, appropriate use of members' task-related skills, and the quality of their task strategy [57]. Specifically, in this study, Minimap was customized so that participants were paired with an ostensible teammate who, in reality, is a pre-programmed agent (referred to as the "teammate"). The teammate and the participant played in the same two-dimensional Minimap environment. This customization of Minimap also enables the investigation of how participants collaborate with their teammates in various ways, including signaling their teammate the position of special victims only the teammate could save, communicating whether a specific location of victims was "cleared", and activating gift boxes that would directly contribute to the bonus of teammates. These observations are then used to build a model to predict how collective intelligence will evolve, allowing agents to collaborate better. Furthermore, autonomous systems can leverage such in-game process observations to make interventions by giving humans advice based on process metrics. As a follow-up research, this observation also points to an opportunity to explore how autonomous agents can adapt to humans as teammates in HAT.

Finally, the use of Minimap is aimed to transcend the scope of individual tasks, provided that microworlds in DDM research literature tend to present a decision-making problem of individuals rather than teams [9, 30]. Minimap is under active development to feature a team environment wherein multiple human players with different roles simultaneously play the game[7]. Moreover, considering that many off-the-shelf machine learning libraries and agent-based models [45] are in Python, Minimap's architecture relying on Python's API framework makes it even easier to incorporate any of these libraries in an attempt to connect and build a synthetic teammate. Given rapid research advancement in HAT, we anticipate the use of Minimap will increase.

5.2 Future work

Minimap has yet to offer a visual interface that allows researchers to design environments and scenario variables interactively, thus still requiring basic programming skills for task customization. Relatedly, Minimap needs to provide a set of diverse scenario templates that researchers can select with a point-and-click interface. Another limitation of this current work is that even though recent research has used Minimap for collaborative tasks in which humans and AI agents are teammates [57], code templates for this type of scenario have yet to be available in this current version of Minimap. We see these limitations as opportunities to improve Minimap's usability and versatility for DDM and AI research in the future.

[7] Multiplayer version of Minimap: `https://github.com/DDM-Lab/team-minimap-ted`

Transparency and Openness

Minimap is provided as a free open-source application, which can be accessed at `https://github.com/DDM-Lab/MinimapInteractiveDDMGame`. All human data and all scripts used for the analyses presented in this manuscript are also available on Github: `https://github.com/DDM-Lab/HumanExperimentMinimap`.

Acknowledgements

This research was sponsored by the Defense Advanced Research Projects Agency and the Army Research Office and was accomplished under Grant Number W911NF-20-1-0006. (DARPA-ASIST program).

References

1. Bamford, C.: Griddly: A platform for ai research in games. Software Impacts **8**, 100066 (2021)
2. Bamford, C., Jiang, M., Samvelyan, M., Rocktäschel, T.: Griddlyjs: A web ide for reinforcement learning. arXiv preprint arXiv:2207.06105 (2022)
3. Beattie, C., Köppe, T., Duéñez-Guzmán, E.A., Leibo, J.Z.: Deepmind lab2d. arXiv preprint arXiv:2011.07027 (2020)
4. Bellemare, M.G., Naddaf, Y., Veness, J., Bowling, M.: The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research **47**, 253–279 (2013)
5. Botvinick, M., Ritter, S., Wang, J.X., Kurth-Nelson, Z., Blundell, C., Hassabis, D.: Reinforcement learning, fast and slow. Trends in cognitive sciences (2019)
6. Brehmer, B.: Feedback delays and control in complex dynamic systems. In: Computer-based management of complex systems, pp. 189–196. Springer (1989)
7. Brehmer, B.: Dynamic decision making: Human control of complex systems. Acta psychologica **81**(3), 211–241 (1992)
8. Brehmer, B.: Feedback delays in complex dynamic decision tasks. Complex problem solving: The European perspective pp. 103–130 (1995)
9. Brehmer, B., Dörner, D.: Experiments with computer-simulated microworlds: Escaping both the narrow straits of the laboratory and the deep blue sea of the field study. Computers in human behavior **9**(2-3), 171–184 (1993)
10. Cegarra, J., Valéry, B., Avril, E., Calmettes, C., Navarro, J.: Openmatb: A multi-attribute task battery promoting task customization, software extensibility and experiment replicability. Behavior research methods **52**(5), 1980–1990 (2020)
11. Chevalier-Boisvert, M., Willems, L., Pal, S.: Minimalistic gridworld environment for gymnasium (2018). URL `https://github.com/Farama-Foundation/Minigrid`
12. Corral, C.C., Tatapudi, K.S., Buchanan, V., Huang, L., Cooke, N.J.: Building a synthetic task environment to support artificial social intelligence research. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 65, pp. 660–664. SAGE Publications Sage CA: Los Angeles, CA (2021)
13. Cronin, M.A., Gonzalez, C., Sterman, J.D.: Why don't well-educated adults understand accumulation? a challenge to researchers, educators, and citizens. Organizational behavior and Human decision Processes **108**(1), 116–130 (2009)
14. Dancy, C.L., Ritter, F.E.: Igt-open: An open-source, computerized version of the iowa gambling task. Behavior research methods **49**(3), 972–978 (2017)
15. De Leeuw, J.R.: jspsych: A javascript library for creating behavioral experiments in a web browser. Behavior research methods **47**(1), 1–12 (2015)

16. Diehl, E., Sterman, J.D.: Effects of feedback complexity on dynamic decision making. Organizational behavior and human decision processes **62**(2), 198–215 (1995)
17. Edwards, W.: Dynamic decision theory and probabilistic information processings. Human factors **4**(2), 59–74 (1962)
18. Elliott, T., Mills, V., et al.: Investigating naturalistic decision making in a simulated microworld: What questions should we ask? Behavior Research Methods **39**(4), 901–910 (2007)
19. Forbus, K.D., Hinrichs, T.R.: Companion cognitive systems: a step toward human-level ai. AI magazine **27**(2), 83–83 (2006)
20. Fothergill, S., Loft, S., Neal, A.: Atc-labadvanced: An air traffic control simulator with realism and control. Behavior Research Methods **41**(1), 118–127 (2009)
21. Freeman, J., Huang, L., Wood, M., Cauffman, S.J.: Evaluating artificial social intelligence in an urban search and rescue task environment. In: AAAI Fall Symposium on Theory of Mind for Teams (2021)
22. Funke, J.: Experimental research on complex problem solving. Complex problem solving: The European perspective pp. 243–268 (1995)
23. Gonzalez, C.: Learning to make decisions in dynamic environments: Effects of time constraints and cognitive abilities. Human Factors **46**(3), 449–460 (2004)
24. Gonzalez, C.: Decision support for real-time, dynamic decision-making tasks. Organizational Behavior and Human Decision Processes **96**(2), 142–154 (2005)
25. Gonzalez, C.: Decision-making: a cognitive science perspective. The Oxford handbook of cognitive science **1**, 1–27 (2017)
26. Gonzalez, C., Dutt, V.: A generic dynamic control task for behavioral research and education. Computers in Human Behavior **27**(5), 1904–1914 (2011)
27. Gonzalez, C., Fakhari, P., Busemeyer, J.: Dynamic decision making: Learning processes and new research directions. Human factors **59**(5), 713–721 (2017)
28. Gonzalez, C., Lerch, J.F., Lebiere, C.: Instance-based learning in dynamic decision making. Cognitive Science **27**(4), 591–635 (2003)
29. Gonzalez, C., Thomas, R.P., Vanyukov, P.: The relationships between cognitive ability and dynamic decision making. Intelligence **33**(2), 169–186 (2005)
30. Gonzalez, C., Vanyukov, P., Martin, M.K.: The use of microworlds to study dynamic decision making. Computers in human behavior **21**(2), 273–286 (2005)
31. Guss, W.H., Houghton, B., Topin, N., Wang, P., Codel, C., Veloso, M., Salakhutdinov, R.: Minerl: A large-scale dataset of minecraft demonstrations. arXiv preprint arXiv:1907.13440 (2019)
32. Henninger, F., Shevchenko, Y., Mertens, U.K., Kieslich, P.J., Hilbig, B.E.: lab. js: A free, open, online study builder. Behavior Research Methods **54**(2), 556–573 (2022)
33. Johnson, M., Hofmann, K., Hutton, T., Bignell, D.: The malmo platform for artificial intelligence experimentation. In: Ijcai, pp. 4246–4247 (2016)
34. Jones, R.E., McNeese, M.D., Connors, E.S., Jefferson Jr, T., Hall Jr, D.L.: A distributed cognition simulation involving homeland security and defense: The development of neocities. In: Proceedings of the human factors and ergonomics society annual meeting, vol. 48, pp. 631–634. SAGE Publications Sage CA: Los Angeles, CA (2004)
35. Kerstholt, J.H., Raaijmakers, J.G.: Decision making in dynamic task environments. In: Decision making, pp. 219–231. Routledge (2002)
36. Klein, G.A.: Sources of power: How people make decisions. MIT press (2017)
37. Lake, B.M., Ullman, T.D., Tenenbaum, J.B., Gershman, S.J.: Building machines that learn and think like people. Behavioral and brain sciences **40** (2017)

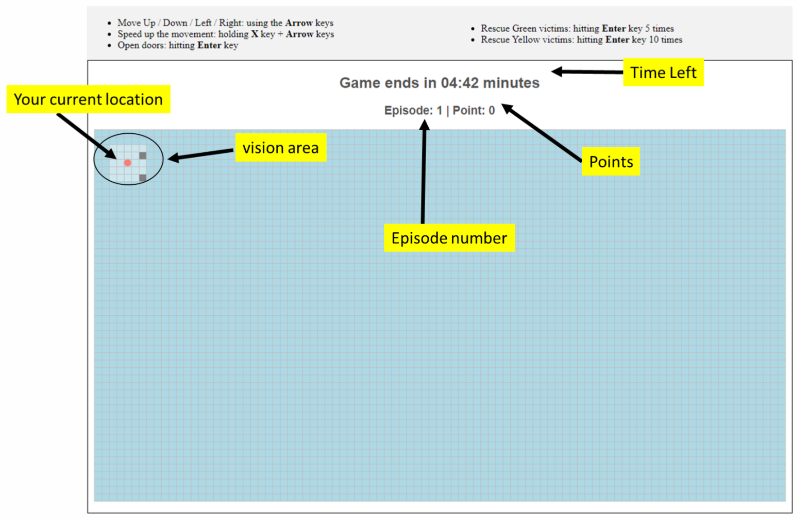
38. Lematta, G.J., Coleman, P.B., Bhatti, S.A., Chiou, E.K., McNeese, N.J., Demir, M., Cooke, N.J.: Developing human-robot team interdependence in a synthetic task environment. In: Proceedings of the Human Factors and Ergonomics Society Annual Meeting, vol. 63, pp. 1503–1507. SAGE Publications Sage CA: Los Angeles, CA (2019)
39. Loft, S., Hill, A., Neal, A., Humphreys, M., Yeo, G.: Atc-lab: An air traffic control simulator for the laboratory. Behavior Research Methods, Instruments, & Computers **36**(2), 331–338 (2004)
40. Madhavan, P., Gonzalez, C., Lacson, F.C.: Differential base rate training influences detection of novel targets in a complex visual inspection task. In: Proceedings of the human factors and ergonomics society annual meeting, vol. 51, pp. 392–396. SAGE Publications Sage CA: Los Angeles, CA (2007)
41. Marusich, L.R., Bakdash, J.Z., Onal, E., Yu, M.S., Schaffer, J., O'Donovan, J., Höllerer, T., Buchler, N., Gonzalez, C.: Effects of information availability on command-and-control decision making: performance, trust, and situation awareness. Human factors **58**(2), 301–321 (2016)
42. McNeese, N.J., Demir, M., Cooke, N.J., Myers, C.: Teaming with a synthetic teammate: Insights into human-autonomy teaming. Human factors **60**(2), 262–273 (2018)
43. Mehlhorn, K., Newell, B.R., Todd, P.M., Lee, M.D., Morgan, K., Braithwaite, V.A., Hausmann, D., Fiedler, K., Gonzalez, C.: Unpacking the exploration–exploitation tradeoff: A synthesis of human and animal literatures. Decision **2**(3), 191 (2015)
44. Nguyen, T.N., Gonzalez, C.: Effects of decision complexity in goal-seeking gridworlds: A comparison of instance-based learning and reinforcement learning agents. In: Proceedings of the 18th intl. conf. on cognitive modelling (2020)
45. Nguyen, T.N., Phan, D.N., Gonzalez, C.: Speedyibl: A comprehensive, precise, and fast implementation of instance-based learning theory. Behavior Research Methods (2022)
46. Omodei, M.M., Wearing, A.J.: The fire chief microworld generating program: An illustration of computer-simulated microworlds as an experimental paradigm for studying complex decision-making behavior. Behavior Research Methods, Instruments, & Computers **27**(3), 303–316 (1995)
47. Osman, M.: Controlling uncertainty: a review of human behavior in complex dynamic environments. Psychological bulletin **136**(1), 65 (2010)
48. O'Neill, T., McNeese, N., Barron, A., Schelble, B.: Human–autonomy teaming: A review and analysis of the empirical literature. Human factors **64**(5), 904–938 (2022)
49. Peirce, J., Gray, J.R., Simpson, S., MacAskill, M., Höchenberger, R., Sogo, H., Kastman, E., Lindeløv, J.K.: Psychopy2: Experiments in behavior made easy. Behavior research methods **51**(1), 195–203 (2019)
50. Peirce, J.W.: Generating stimuli for neuroscience using psychopy. Frontiers in neuroinformatics **2**, 10 (2009)
51. Rafols, E.J., Ring, M.B., Sutton, R.S., Tanner, B.: Using predictive representations to improve generalization in reinforcement learning. In: IJCAI, pp. 835–840 (2005)
52. Schelble, B.G., Flathmann, C., McNeese, N.J., Freeman, G., Mallick, R.: Let's think together! assessing shared mental models, performance, and trust in human-agent teams. Proceedings of the ACM on Human-Computer Interaction **6**(GROUP), 1–29 (2022)
53. Schmid, U., Ragni, M., Gonzalez, C., Funke, J.: The challenge of complexity for cognitive systems (2011)
54. Sterman, J.D.: Misperceptions of feedback in dynamic decision making. Organizational behavior and human decision processes **43**(3), 301–335 (1989)
55. Stoet, G.: Psytoolkit: A novel web-based method for running online questionnaires and reaction-time experiments. Teaching of Psychology **44**(1), 24–31 (2017)
56. Todd, P.M., Hills, T.T., Robbins, T.W.: Cognitive search: Evolution, algorithms, and the brain. MIT press (2012)
57. Zhao, M., Eadeh, F.R., Nguyen, T.N., Gupta, P., Admoni, H., Gonzalez, C., Woolley, A.W.: Teaching agents to understand teamwork: Evaluating and predicting collective intelligence as a latent variable via hidden markov models. Computers in Human Behavior p. 107524 (2022)

Appendices

A Minimap: Generic Instructions

“Minimap” rescue mission: Structure of the game and Goals

In this task your goal is to rescue as many victims as possible from inside a building within a time limit. The Figure shows the interface of this game.



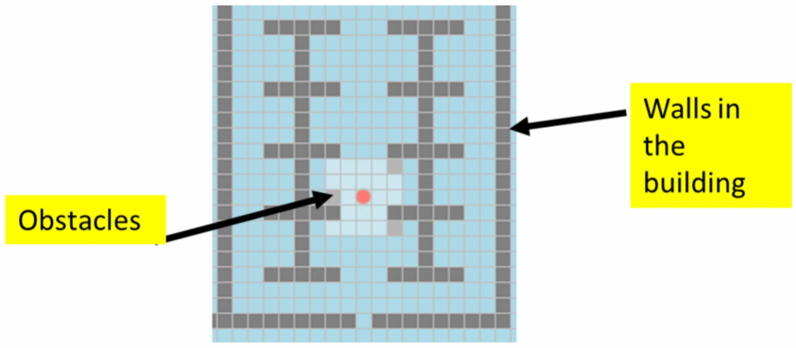
- Move Up / Down / Left / Right: using the **Arrow** keys
- Speed up the movement: holding **X** key + **Arrow** keys
- Open doors: hitting **Enter** key

- Rescue Green victims: hitting **Enter** key 5 times
- Rescue Yellow victims: hitting **Enter** key 10 times

Game ends in 04:42 minutes

Episode: 1 | Point: 0

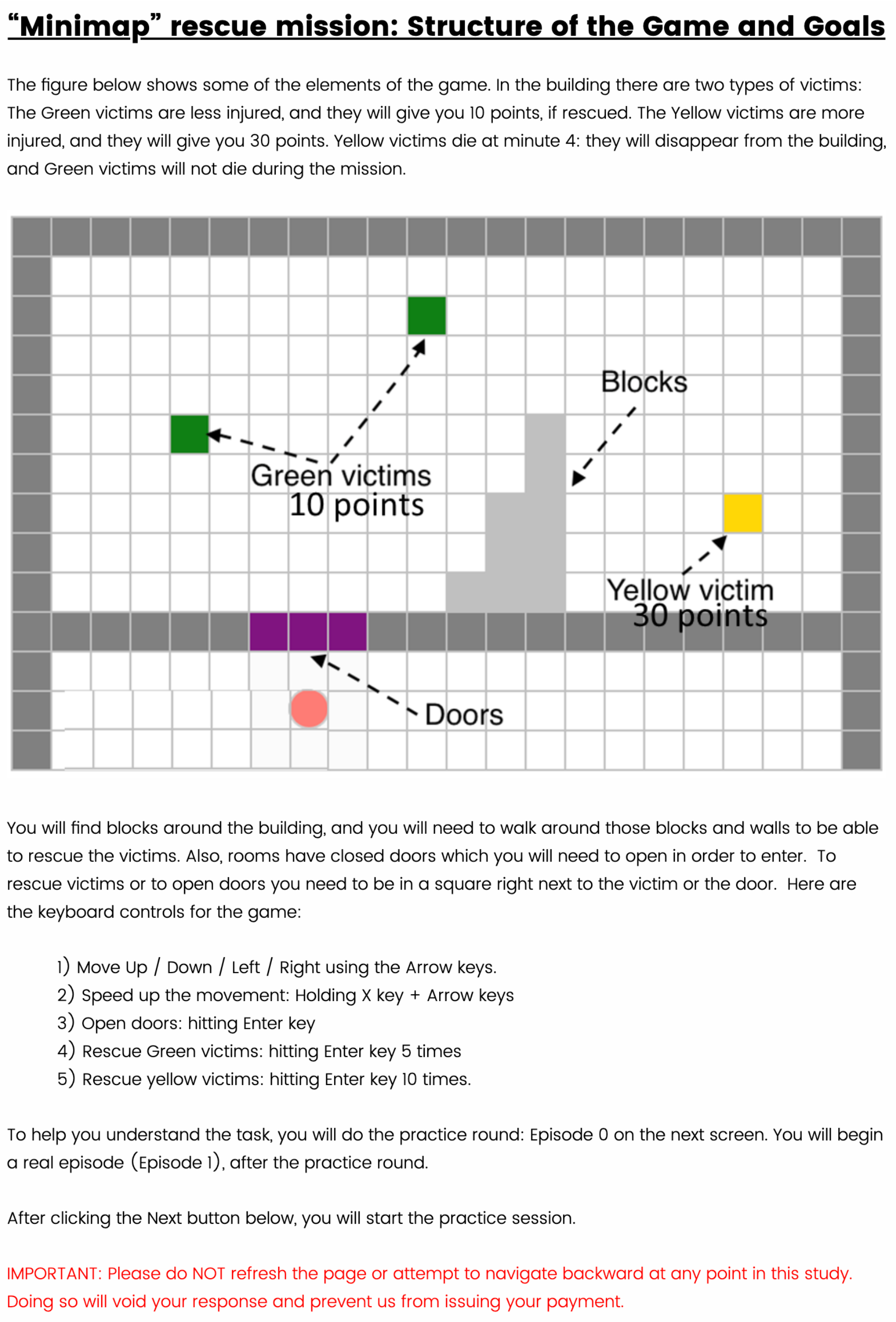
Your location within the building is identified as a pink dot on the screen. The building has many rooms, doors, walls and obstacles around. Sometimes, you will only be able to see the walls and obstacles around you as shown in this section of the building:



but other times the walls and the obstacles will be concealed, and you will only be able to see an area of 5x5 squares around you (“vision area”). Thus, you need to search around to find the victims to rescue, which are identified as Green and Yellow squares.

You will perform a training scenario (i.e., Episode: 0), and 8 real episodes where you can earn points. The training episode lasts 1 minute, and each of the real episodes (Episode 1… 8) last 5 minutes. The time left in the episode will be shown in the screen (i.e., Time Left). You will earn points by rescuing the victims and these points accumulate (i.e., “Points”) during each of the real episodes. Your bonus will be calculated in the following way. The total points earned across all 8 episodes will be divided by 8, and we will pay 1 cent for each transformed point. For example, if you earned 2800 points during the 8 episodes, that divided by 8 = 350, and 1 cent per adjusted point gives us $3.50.

Fig. A1: Instructions on the structure and objective of the game



“Minimap” rescue mission: Structure of the Game and Goals

The figure below shows some of the elements of the game. In the building there are two types of victims: The Green victims are less injured, and they will give you 10 points, if rescued. The Yellow victims are more injured, and they will give you 30 points. Yellow victims die at minute 4: they will disappear from the building, and Green victims will not die during the mission.

You will find blocks around the building, and you will need to walk around those blocks and walls to be able to rescue the victims. Also, rooms have closed doors which you will need to open in order to enter. To rescue victims or to open doors you need to be in a square right next to the victim or the door. Here are the keyboard controls for the game:

1) Move Up / Down / Left / Right using the Arrow keys.
2) Speed up the movement: Holding X key + Arrow keys
3) Open doors: hitting Enter key
4) Rescue Green victims: hitting Enter key 5 times
5) Rescue yellow victims: hitting Enter key 10 times.

To help you understand the task, you will do the practice round: Episode 0 on the next screen. You will begin a real episode (Episode 1), after the practice round.

After clicking the Next button below, you will start the practice session.

IMPORTANT: Please do NOT refresh the page or attempt to navigate backward at any point in this study. Doing so will void your response and prevent us from issuing your payment.

Fig. A2: Instructions on how to use the controls to play the game.

B Customizaing Experiments with Minimap

This section presents how to set up an experiment with Minimap as an introductory tutorial and web service API methods in the current version of Minimap.

B.1 Preparation

Running the Minimap server requires `MySQL` database.

– Install `MySQL` for your operating system (Win/Mac/Linux)

- Check that `MySQL` are installed. Open a terminal or a command prompt and run:

```
mysql --version
```

- Connect to the `MySQL` server, run

```
mysql -u user -p
```

  `user` represents the user name of your MySQL account. Substitute appropriate values for your setup. MySQL will display a prompt to enter the corresponding password.

- From the `mysql` prompt, create a database:

```
CREATE DATABASE <db_name>;
```

- In a command shell, import the SQL script named `script.sql` provided in the project folder

```
mysql -u user -p db_name < script.sql
```

B.2 Installation/Quick start

- In a command shell, go to the main folder that contains the `requirements.txt` file.
- Create a virtual Python Environment is recommended by running the following commands in your shell. (This may be different on your machine. If it does not work, look at how to install a virtual python env based on your system):

```
python3 -m venv env
source env/bin/activate
```

- Install the required python libraries by running this command in your shell:

```
pip install -r requirements.txt
```

- In the file `mission/db.py`, update the database configuration according to the database setup in the local machine, as in the following example:

```
user = "root"
password = ""
host = "localhost:3306"
db_name = "minimap_visibility_complex"
```

- To start the server, run:

```
python server.py
```

- While the server is running, navigate to: `http://0.0.0.0:5700`
- To stop the server, run `Ctrl+C`

B.3 Running an experiment with Minimap

– After the server is started, launch the default search and rescue mission by connecting to:

```
http://0.0.0.0:5700/minimap?uid=XX
```

The port number can be configured in the `server.py` file.

– To check the replay interface, visit:

```
http://0.0.0.0:5700/replay
```

– Web service APIs for getting information about the playing score and the current game episode:

```
GET "/score/{uid}/{condition}"
GET "/episode/{uid}/{condition}"
```

B.4 Directory structure of an experiment

The experiment folder called `mission` contains the following fundamental sub-folders and files:

```
main folder
├── mission
│   ├── static
│   │   ├── data
│   │   ├── js
│   │   └── style.css
│   ├── templates
│   ├── config.json
│   ├── db.py
│   └── main.py
├── requirements.txt
├── script.sql
└── server.py
```

The `mission/static` folder contains all the static files the game will serve. In particular, the `mission/static/data` folder includes all the base or new map template designs, while the `mission/static/js` folder compriese of all the files defining the behavior of the game. The `mission/templates` folder contains all the `html` files that define the game's interface.

The database configuration is declared in the `mission/db.py` file while `mission/main.py` file controls the web service APIs available for use. Importantly, as described earlier, all essential DDM features are fully customizable in Minimap. One can configure the experiment settings by changing the appropriate values in the `mission/config.json` file.

B.5 User modifiable configuration parameters

We now describe how experimenters can customize the game setting through Minimap configuration to fit the scientific requirements of the scientific endeavor.

The experiment settings are specified in the file `mission/config.json`. This file contains the variables needed to define the functioning of the experiment as follows:

```
{
    "cond_name": "complex-full",
    "load_new_map": 1,
    "map_file": "map_design.csv",
    "map_dir": "mission/static/data/",
    "tile_width": 13,
    "game_duration": 300,
    "max_episode": 8,
    "victim_die": 120,
    "visibility": "full",
    "complexity": "complex",
    "delayed_time": 1000,
    "perturbation_time": 120,
    "map_perturbation": "map_perturbation.csv"
}
```

Condition Name. As it is likely that an experiment will be run under multiple conditions, the parameter `‘‘cond_name’’` is used to identify each experimental condition. One can change the default condition name, and make it short and descriptive.

Interface Modification. Minimap, by default, features the environment as illustrated in Fig. 1 provided that the environment design is customizable. To create different environments, one can customize the Minimap layouts, including the location and the number of objects on the map (e.g., victims, walls, obstacles, and doors). Technically, this can be achieved without requiring specialized technical knowledge: modifying the base map designed in the Excel file, placed in the folder (`mission/static/map_design.xlsx`), and then saving it as a CSV file. Afterward, it is needed to specify in the `config.json` file that the newly designed layout is used for the game (`‘‘load_new_map’’: 1`, 0 otherwise) together with the name of the new map in CSV configuration (`‘‘map_file’’: ‘‘map_design.csv’’`). The width of each tile/cell in the gridworld is set by the variable `‘‘tile_width’’`. One can increase or decrease this value to adjust the size of the map. Detailed instructions on how to modify a task interface are presented in B.7.

Time Constraints. The variable `‘‘game_duration’’` controls the duration of playing time within an episode. The value is in seconds, so for a 5-minute mission, the duration is set to 300.

Complexity. Minimap provides two levels of structural complexity: *simple* and *complex*. For example, the configuration file above makes the game environment complex with (`‘‘complexity’’: ‘‘complex’’`). Structural complexity is determined based on the number of obstacles and the arrangement of critical victims (see Fig. 2 and Section 3.2.2 for more details). One can freely create new maps with different layouts and numbers of objects by following the steps described in the interface modification section.

Visibility. Minimap includes three levels of visibility to manipulate, namely the field of view (*fov*), map view (*map*), and full view (*full*). This degree of visibility is specified by `‘‘visibility’’` in the `config.json` file. With a field of view, the individual can only observe part of the environment within their restricted vision area. In contrast, with a map view, they can see the overview structure of the map in addition to their vision area (Fig. 3) for more details). Conversely, the full view gives the player a complete picture of the environment, including where the victims are.

Time Victims Die. The moment victims die in the search and rescue task can be varied through the variable `‘‘victim_die’’`. For example, in the above configuration file, critical victims die or disappear from the building after two minutes (120 seconds) of the game.

Delayed Time. The delay in the individual’s actions, associated with the dynamic complexity, is integrated into Minimap by the variable `‘‘delayed_time’’`, measured in seconds. The affected activities include victim-saving and door-opening. In the example above, the delay period is set to one second, meaning that after the player executes an action of rescuing a victim, it takes one second for the effort to take effect. The action occurs immediately with a delayed time of zero.

Perturbation Time. The variable `‘‘perturbation_time’’` controls when a perturbation or a sudden change within an episode occurs measured in seconds. For example, during a 5-minute mission, if the perturbation takes place after 2 minutes of gameplay, then the value to 120.

Perturbation Map. The variable `‘‘perturbation_map’’` specifies the file associated with the new scenario layout once a perturbation occurs. The file is in `CSV` format as in the map design file.

Number of Playing Episodes. The number of times an individual can perform the task is a variable in Minimap. For example, `‘‘max_episode’’` in the `config.json` file above controls the number of playing episodes `‘‘max_episode’’: 8` i.e., the default value is 8). This feature allows experimenters to run experiments multiple times to examine the human learning process.

B.6 Reproducing the experiments on complexity and opaqueness with Minimap

To reproduce each experimental condition described in Section 4, one can configure the corresponding parameters in the `config.json` file under folder `mission` as follows.

Condition 1. Simple FoV:

```json
{
    "cond_name": "simple-fov",
    "load_new_map": 0,
    "tile_width": 13,
    "game_duration": 300,
    "max_episode": 8,
    "victim_die": 240,
    "visibility": "fov",
    "complexity": "simple",
    "delayed_time": 0,
    "perturbation_time": 0,
}
```

Condition 2. Simple Map:

```json
{
    "cond_name": "simple-map",
    "load_new_map": 0,
    "tile_width": 13,
    "game_duration": 300,
    "max_episode": 8,
    "victim_die": 240,
    "visibility": "map",
    "complexity": "simple",
    "delayed_time": 0,
    "perturbation_time": 0,
}
```

Condition 3. Complexity FoV:

```json
{
    "cond_name": "complex-fov",
    "load_new_map": 0,
    "tile_width": 13,
    "game_duration": 300,
    "max_episode": 8,
    "victim_die": 240,
    "visibility": "fov",
    "complexity": "complex",
    "delayed_time": 0,
    "perturbation_time": 0,
}
```

Condition 4. Complexity Map:

```
{
    "cond_name": "complex-map",
    "load_new_map": 0,
    "tile_width": 13,
    "game_duration": 300,
    "max_episode": 8,
    "victim_die": 240,
    "visibility": "map",
    "complexity": "complex",
    "delayed_time": 0,
    "perturbation_time": 0,
}
```

Start the server and launch the new task scenario by connecting to:

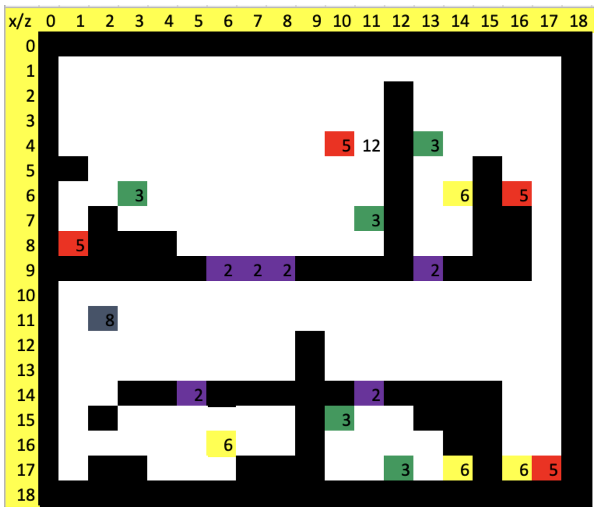
```
    http://0.0.0.0:5700/minimap?uid=XX
```

B.7 An example of creating a task scenario

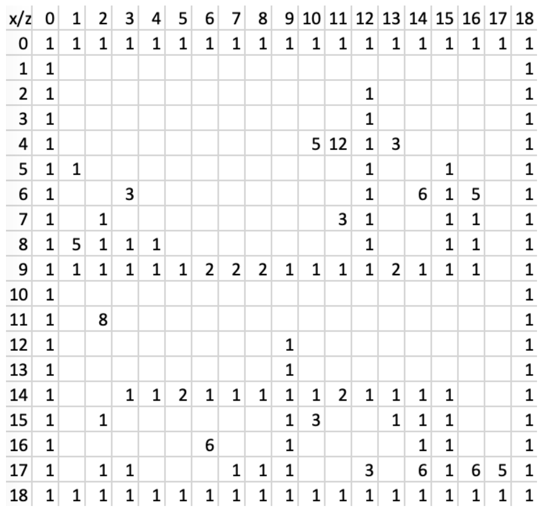
This section describes a step-by-step guide for creating a task scenario in Minimap. Suppose that in this new scenario, a player is instructed to navigate the 19×19 environment to save three types of victims: critical (red) victims, serious (yellow), and minor (green) victims. The player needs to play 10 episodes of a 3-minute game. A perturbation occurs after one minute of the game, which entails the sudden appearance of rubble and an increased number of serious and critical victims. The serious and critical victims vanish when two minutes elapse.

B.7.1 ***Step 1: Creating the environment layout***

We create a map design file in the `Excel` configuration (e.g., `‘‘map_design_c.xlsx’’` as illustrated in Fig. A3a). Afterward, we save it in the `CSV` (`‘‘map_design_c.csv’’`) and place it in the folder `mission/static/` (see Fig. A3b).



(a) Map design in an `Excel` file



x/z	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1																		1
2	1												1						1
3	1												1						1
4	1										5	12	1	3					1
5	1	1											1			1			1
6	1			3									1		6	1	5		1
7	1		1									3	1			1	1		1
8	1	5	1	1	1								1			1	1		1
9	1	1	1	1	1	1	2	2	2	1	1	1	1	2	1	1	1		1
10	1																		1
11	1		8																1
12	1									1									1
13	1									1									1
14	1			1	1	2	1	1	1	1	1	2	1	1	1	1			1
15	1		1							1	3			1	1	1			1
16	1						6			1					1	1			1
17	1		1	1				1	1	1			3		6	1	6	5	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(b) Map design in a `CSV` format

Fig. A3: Illustration of map design (in the `Excel` and `CSV` format) for the new scenario

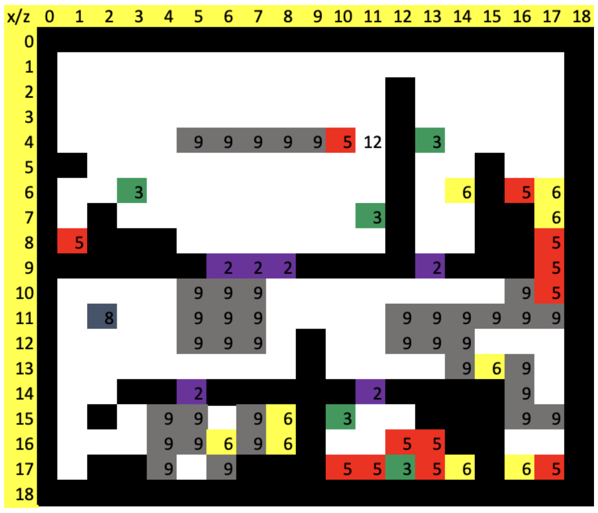
The code number of each object in the environment, together with their associated action and reward, is described in Table A1. One can easily adjust the value of rewards and/or the number of button presses needed to obtain each object. The reward for obtaining each object and the number of keystrokes required for such action is defined in the column “Press” and “Reward”. For instance, change the values of the two columns “Press” and “Reward” in the third row of the table to adjust the number of key presses needed and the reward received by saving the green objects.

Table A1: Object codebook of Minimap. The column “Press” determines the number of required keystrokes to obtain the object, while the column “Reward” dictates the received reward for each object.

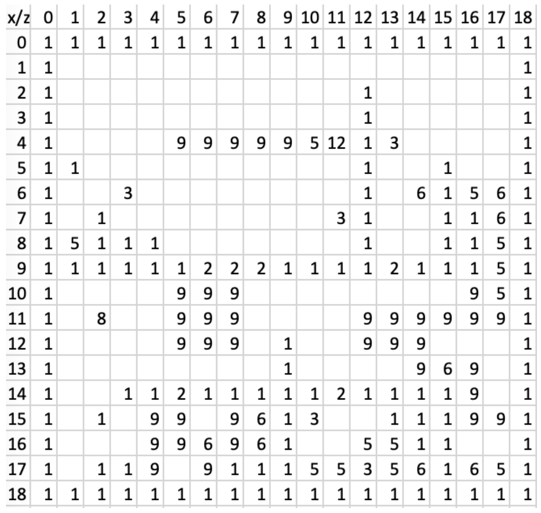
Num	Name	Press	Reward
1	wall	0	0
2	door	1	0
3	green	5	10
4	blue	10	15
5	red	20	60
6	yellow	10	30
7	other	0	0
8	agent	0	0
9	rubble	5	0

B.7.2 Step 2: Adding perturbations

We can intensify the task dynamic with sudden perturbation in the middle of the gameplay (within an episode). To do that, we design a new setting when the perturbation happens in the `Excel` file. When one minute elapses, the player will suddenly experience the appearance of rubble and an increase in critical and serious victims. We name the file, e.g., ‘‘`map_perturbation_c.xlsx`’’ as illustrated in Fig. A4a. We also save the design file in the `CSV` format and place it in the folder `mission/static/` (see Fig. A4b).



(a) Perturbation design in an `Excel` file



x/z	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1																		1
2	1												1						1
3	1												1						1
4	1					9	9	9	9	9	5	12	1	3					1
5	1	1											1			1			1
6	1			3									1		6	1	5	6	1
7	1		1									3	1			1	1	6	1
8	1	5	1	1	1								1			1	1	5	1
9	1	1	1	1	1	1	2	2	2	1	1	1	1	2	1	1	1	5	1
10	1					9	9	9									9	5	1
11	1		8			9	9	9					9	9	9	9	9	9	1
12	1					9	9	9		1			9	9	9				1
13	1									1					9	6	9		1
14	1			1	1	2	1	1	1	1	1	2	1	1	1	1	9		1
15	1		1		9	9		9	6	1	3			1	1	1	9	9	1
16	1				9	9	6	9	6	1			5	5	1	1			1
17	1		1	1	9		9	1	1	1	5	5	3	5	6	1	6	5	1
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

(b) Perturbation design in a `CSV` format

Fig. A4: Illustration of perturbation design (in the `Excel` and `CSV` format) for the new scenario

B.7.3 Step 3: Configurating experiment parameters

For the task described above, we adjust the file `config.json` in the folder `mission` as the following:

```
{   "load_new_map": 1,
    "map_file": "map_design_c.csv",
    "map_dir": "mission/static/data/",
    "tile_width": 20,
    "game_duration": 180,
    "max_episode": 10,
    "victim_die": 120,
    "visibility": "full",
    "delayed_time": 1000,
    "perturbation_time": 60,
    "map_perturbation": "map_perturbation_c.csv"}
```

Since we use the newly designed interface for the task, the value of ``load_new_map'' variable is set to 1 (true). We also increase the value of tile_width variable to 20 to expand the size of the grid environment. By default, the visibility (opaqueness) is set to full view (``full''). We can assign the value of the variable visibility to ``fov'' or ``map'' for displaying the task with the field of view or map structure, respectively.

B.7.4 ***Step 4: Running the task***

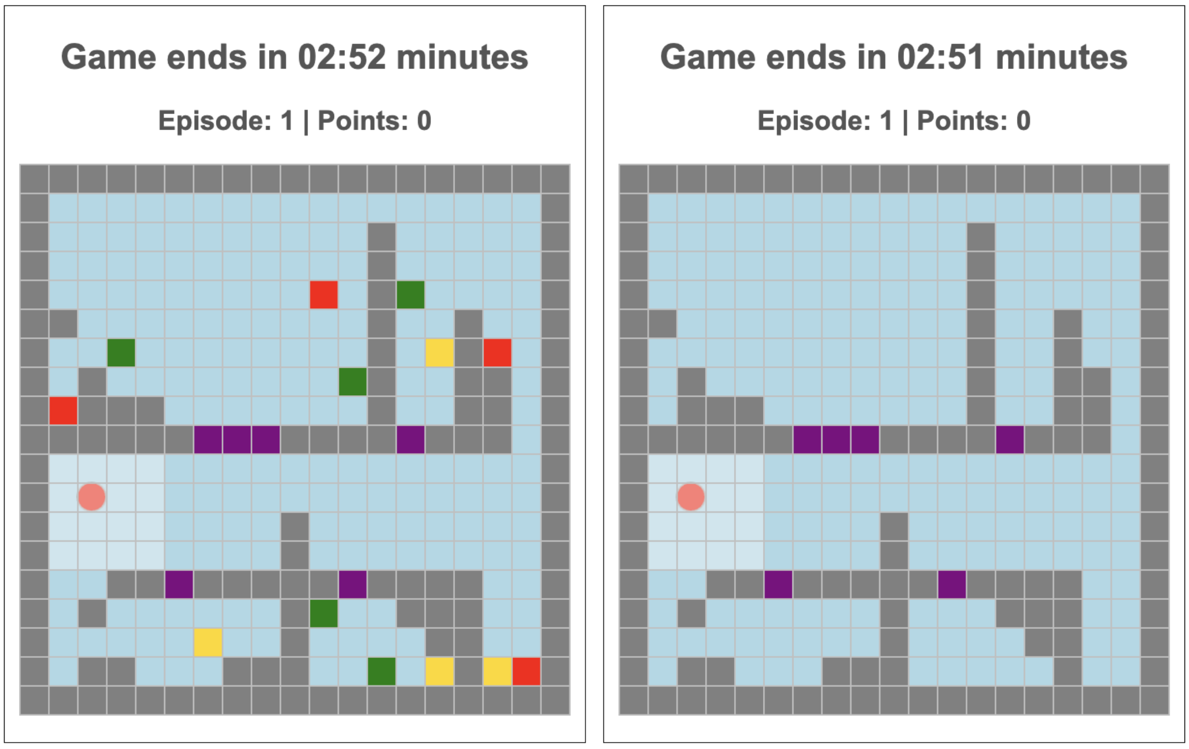
We start the server by typing:

```
python server.py
```

Once the server is started, launch the new task scenario by connecting to:

```
http://0.0.0.0:5700/minimap?uid=XX
```

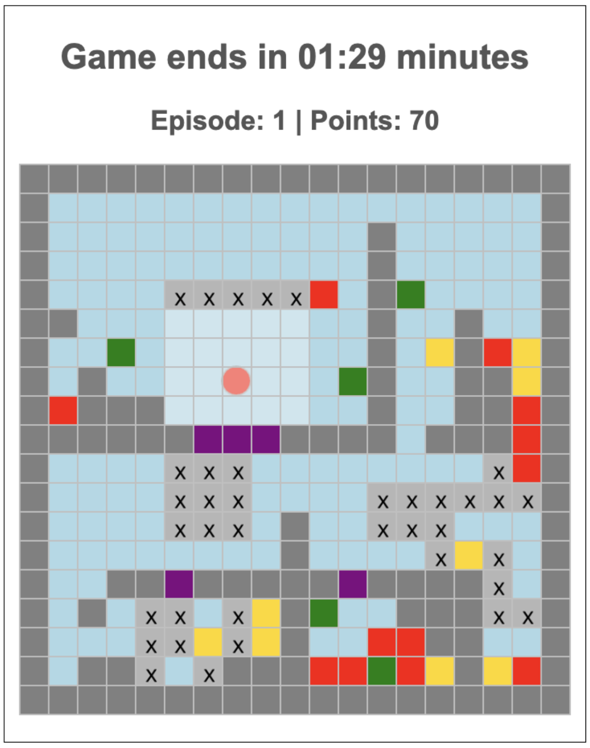
The interface of the new task with the full view and map view is illustrated in Fig. A6.



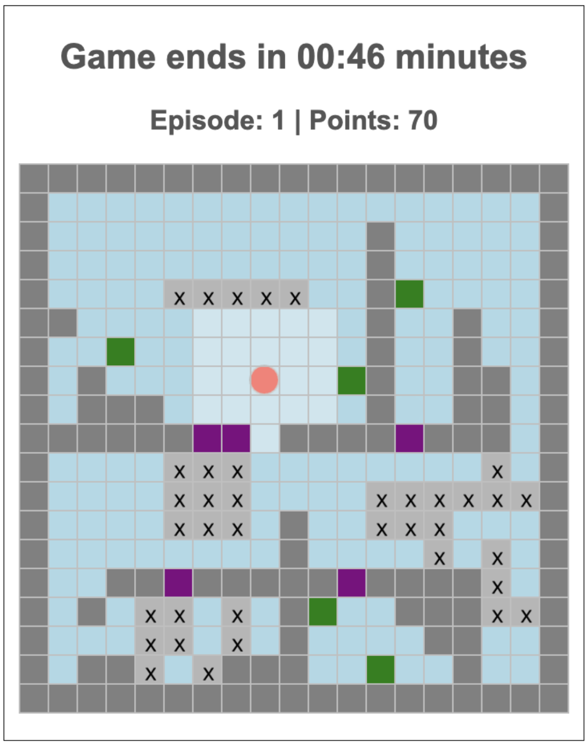
(a) New task with full view (``full'') (b) New task with map view (``map'')

Fig. A5: Display of the new task scenario in Minimap

Fig. A6a illustrates the environment when the disruption happens (i.e., rubble appears, and the number of critical (red) and serious (yellow) victims grows). Fig. A6b shows the setting after the time in ``victim_die'' variable elapses (i.e., the critical and serious victims disappear).



(a) Environment when the perturbation happens



(b) Environment after the critical and serious victim vanish

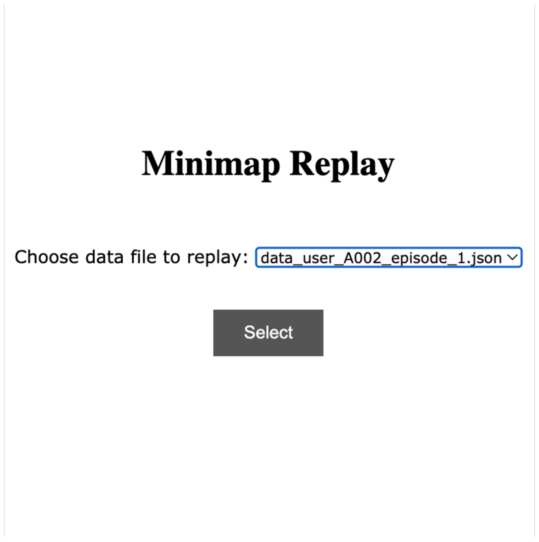
Fig. A6: Illustration of the task environment when the disruptions take place.

A-.7.5 ***Step 5: Replaying recorded data files***

We can replay the recorded data files by entering

```
http://0.0.0.0:5700/replay
```

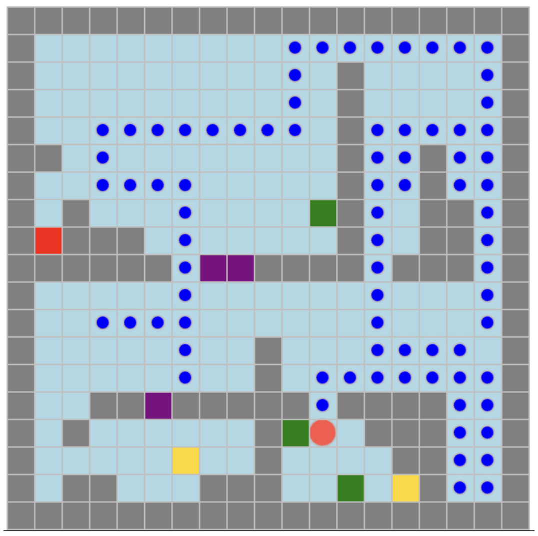
Then select the data file to display the playback illustrated in Fig A7.


Minimap Replay

Choose data file to replay: data_user_A002_episode_1.json

Select

(a) Interface of selecting the data file for the playback



(b) Illustration of the replay with the new task

Fig. A7: Illustration of the replay function with the new task scenario