BioPlan: Classical Planning with Crowd simulation

Maurício C. Magnaguagno*
and Felipe Meneguzzi†
Pontifícia Universidade Católica do Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil
Emails: *mauricio.magnaguagno@acad.pucrs.br, †felipe.meneguzzi@pucrs.br

***Abstract*—Crowd simulation for evacuation situations often assumes that all agents are trying to reach a single point within an environment. Although such an assumption is not entirely wrong, human agents often exhibit more complex behaviors, even if deviations from the standard behavior are not particularly frequent. Classical planning is far from the best way to achieve the minimal path or correct behavior for agents, but adds a deeper level of reasoning about complex goal-achievement and about actions that are more complex than simply moving about. In this paper, we describe a crowd simulation experiment that uses classical AI planning to enrich the behavior of agents in the scenario. Using this approach, we can express not only the target destination of agents, but also (sub)goals and path preferences.**

I. Introduction

Computer games and simulations are often concerned with virtual crowds to populate their simulated environments, with each specific application focusing on different concerns. Whereas games are generally concerned with computational efficiency aimed at achieving a responsive experience, even at the cost of some of its realism, simulations are concerned with realistic virtual crowds that respond in a way that is compatible with real humans. Common to all of these applications, the most basic problem to be solved is to make individual agents navigate through a set of waypoints that are either dynamically generated or previously established by a designer. However, when an agent have more complex goals than simply arriving at one or more destinations, the problem becomes one in which an agent needs to achieve additional (sub)goals which need to be exhibited in the simulation. To achieve these types of goals without having to specify, at design-time, exactly the actions to be taken by an agent, a planner is often required. In fact, AI planners have been extensively used to compute the behavior of individual agents in computer games [1]. However, computing individual plans for large numbers of agents with many possible actions is prohibitively time-consuming, which leads to the issue of using planning algorithms efficiently to generate behavior for several similar agents.

In this paper, we describe a crowd simulation approach that uses classical AI planning to enrich the behavior of agents through a path influenced by agent desires shared by groups of agents. Using this approach, we can express not only the target destination for the agents, but also (sub)goals as desired states to be reached, including parts of the map as path preferences. Our approach consists of converting agent preferences to a classical planning problem and employing a classical planner before the simulation to generate paths with subgoals for agents within a crowd to follow once the simulation starts. Although deterministic plans from classical planners are often not suitable for generating behavior in complex spatial environments, we use the higher level of abstraction of such plans in a way that allows rich crowd behavior without a substantial addition in complexity to the simulator. We achieve this by letting the collision avoidance mechanism already present in the simulation add uncertainty to the resulting plan-driven behavior of the agents. This makes the solutions more flexible, with agents always looking for alternative ways to reach their desires while maintaining a least-effort path.

Traditional approaches to crowd simulation have avoided the issue of planning towards global behaviors more complex than moving agents from one point in a map to another, thus focusing reasoning only on the path itself [2]. This type of reasoning prevents agents from pursuing desired states other than map positions. Such a choice is motivated by the fine-grained representation of the simulated maps, which avoids discretizing the space into a coarse grid to achieve a realistic representation of individual agent movement. Using such a state representation, reasoning about agent desired-states becomes infeasible, since scalability issues arise once thousands of agents have a single location as their desired destination. Treuille et al. [3] points out that global planning with local collision avoidance may lead to unrealistic situations such as large crowd concentrations with agents feeding into a congested mass of agents, whereas real humans would try to avoid the crowded region before getting stuck in it by blindly following their path. In response, the authors of *Continuum Crowds* have developed a method to plan globally so that no agent ever gets stuck in the environment. Alternatively, Li et al. [4] try to achieve complex behavior while minimizing the global planning effort by employing the notion of a group leader that performs complex reasoning, while a crowd of agents conceptually follows this leader through a series of checkpoints as they move through the scenario. In this setting, the leader not only plans for itself, but also needs to make sure that its behavior includes tolerances so that the crowd of followers does not get stuck. By contrast, our approach uses a classical planner to generate plans for a group of agents in a single execution, generating a single plan that is executed by each individual agent in a group and letting the existing collision avoidance algorithms deal with the details of movement at runtime. Although the use of planning capabilities to expand behavior has been used previously as a cognitive layer for animated characters [5], to the best of our knowledge, its efficient use to drive behavior in crowd simulation is novel.

Thus, our main contribution is an approach that uses classical planning to efficiently enrich a traditional crowd simulation model with the notion of declarative goals [6] while maintaining a substantial degree of scalability. We demonstrate

the applicability of our approach through an implementation based on the BioCrowds simulator [7] and a classical planner that does STRIPS-style planning [8] using heuristic search [9]. The resulting system simplifies the effort of crowd design and control while speeding up the computation of multiple group paths by automatically generating the data required by the planner whenever new paths are required. In this article, we extend our original work [10] in two main ways. First we include an automatic mechanism to determine which positions are important to be reached, to allow the generated plans to be relaxed without restrictions. Second, we add different views of the same map for each group to create different levels of access, restricting agents to use different parts of the map. Such extensions allow more realistic scenarios to be simulated without removing designer control.

II. Background

Path planning[1] algorithms are commonly used within simulations to compute paths from a starting position to a goal position for agents to use according to their behaviors. Thus, in this section we review relevant prior work on path planning and agent behavior. One of the key aspects of a crowd simulation is the way in which agents move through the simulated space and interact with each other when collisions may occur. One of the first crowd simulation systems, Boids [11], uses a behavior model with simple rules to generate actions based on agent perception. These rules controlled attraction and angle of movement of bird-like creatures and later evolved to more human-like features [12]. By contrast, we focus on the model used by the BioCrowds simulator [7], which is based on a biological phenomenon, which compares the growth of veins in leaves with the social interaction of agents competing for space while using trajectory guides.

Although guiding an agent's path using predetermined tasks (sequences of actions to be executed) is not new in BioCrowds [12], its agents' behavior is limited to a single destination during the simulation. The model proposed by [12] selects a random action to be performed and a path planning algorithm computes a path to the point where the action is possible. Thus, BioCrowds agents lack the ability to reason about higher-level goals (world-states to be achieved) and multiple goals and subgoals. In order to address this limitation, we aim to use a more advanced reasoning mechanism, whereby agents have a desired world-state and use a domain-independent planning algorithm to compute a sequence of complex actions in order to transition from the current world-state into the desired world-state. Here, a world-state may refer not only to a position in the environment, but also other, more abstract goals such as avoiding a certain position.

Planning algorithms search using a specification of the environment dynamics using transition rules described in a flexible formal language as well as a specification of the problem to be solved [13]. This allows very different problems and domains to be solved by the same, efficient, algorithm. On the other hand, path finding problems are usually solved by tailored search algorithms, which generate a plan with a good response time but much less flexibility. For example, adding keys and closed doors to an environment requires the reconstruction of the entire search algorithm instead of a minimal reformulation of the domain to add actions to pick keys and unlock doors. The key point of planning is reusability, and planning research has yielded a number of formal languages, such as the well-known *Stanford Research Institute Problem Solver* (STRIPS) language [8], and more recent the formalizations of the *Planning Domain Definition Language* (PDDL), which is the standard planning language [14] for the ICAPS competition [15]. This is analogous to the notions of procedural and declarative goals known in the autonomous agents literature [6], [16], which we borrow in this work. The usual approach is to use a procedural goal, where a predetermined procedure once executed successfully will achieve the goal. Our approach is based on declarative goals, where the desired state is declared and the plan is not readily available as a procedure, which requires a classical planner to find the sequence that achieve the goal-state.

[1]Path planning is often referred to simply as planning in crowd simulation literature as in [3], in this paper, to avoid confusion, we shall differentiate path planning from AI planning (or simply planning).

III. Crowds

Several approaches have been used to model crowd behavior. The most influential models are based on such different approaches as: flocking behavior [11], sociological factors [17], psychological effects [18], geographically-based direction [19] and social forces [20]. The models are usually concerned with local problems while some global path planning is used to compute plans to reach an agent's goals. Underlying all of these approaches, is a particle system called flocking model [11], a set of small sprites with movement based on rules, called steering behaviors, used to create a consistent and fluid movement using local perception. All of these approaches use the following three fundamental steering behavior rules:

- **Separation**: avoid crowding near particles, particles try to keep safe from collision in dense regions;
- **Alignment**: follow the average heading angle of near particles to keep itself as part of the crowd going to the same place; and
- **Cohesion**: move towards the center of near particles, keeping particles together.

This basic set of rules can be expanded to create realism for specific situations and recreates the idea of an unlimited population for the crowd, as flocks emerge naturally with each agent sensing only agents nearby.

A. BioCrowds

The BioCrowds model is based on a biological phenomenon that happens in leaves, where leaf veins are attracted to the auxin hormone, resulting in a particular growth patterns. Different parameters of this distribution and attraction make different veins possible, giving each plant a unique pattern. Since the veins compete for the auxins, the model resembles a crowd where each agent competes for more individual space, aiming to achieve a collision-avoidance system between agents in a dense crowd. The behaviors of crowds being simulated must be checked at runtime, to ensure the following patterns are present:

- **lane formation**: as agents move, the free space behind them becomes attractive for the other agents that are

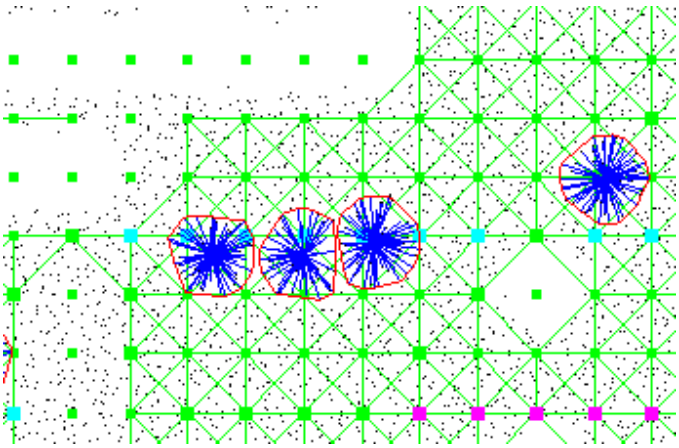

Fig. 1. BioCrowds

going in the same direction, inducing several agents to use such spaces as a collision-free path, limited only by the speed of the agents ahead;

- **arch formation**: the flow of agents around an obstacle creates an arch around the obstacle as agents avoid colliding with it — arches can be seen as a special case of a lane;
- **speed based on density**: as more agents occupy the same region their speed is affected to avoid collision with other agents;
- **bottlenecks at small spaces**: as narrow paths are used by many agents, the competition for the passage causes agents to actually wait for the ones ahead of them to free the passage — bottlenecks may appear without walls around them if a resource is the destination of several agents; and
- **divergence**: after passing through a bottleneck, agents usually follow different paths, creating the inverse of bottlenecks, as no obstacles or agents are there to slow down their movement.

Those patterns did emerge in BioCrowds, showing that the model actually resembles the human interaction that occur in crowded spaces [21]. Agents compete for fixed markers (such as the auxins) distributed randomly across the free space, allocating and freeing these markers as they move across the space, allowing other agents to occupy their previous space. The amount of markers allocated around each agent is based on their requirement for personal space. This way of reasoning about space is called proxemics, which define different types of relation agents have based on distance to other elements in the environment. Proxemics dictate the distance agents try to keep from each other, simplifying the obstacle reasoning required, as positions without free markers are considered impossible to occupy.

IV. Classical AI Planning

Classical planning is an area of AI concerned with creating algorithms to solve problems defined with a generic formal language that treat environment states as sets of discrete variables [13]. A classical planner is usually based on a search algorithm that tries to find a sequence of actions (formally defined as a plan) that, when executed, modifies the initial state of the world into the desired goal state. This plan is the sequence of intermediary points obtained from the usual A^* for path planning, but instead of points, the effects of each action yield the desired state when executed in order and successfully. Below, we summarize the key concepts in AI planning required for our work:

- **free-variables** are placeholders for any object in a problem, and which are used in action schemas to define a large number of ground actions;
- a **predicate** is a named property of the world with any number of terms, each term can be a free-variable or an object;
- **objects** are explicitly or implicitly defined by the problem — once defined a problem's actions can be expanded into its possible instances;
- a **proposition** is a ground predicate, in which each term is an object;
- a **state** is a structure containing world properties as propositions at a particular point in time;
- an **action** or **operator** is part of a domain's transition function that can be applied to the current state, it is specified in terms of preconditions and effects (expressed as logical formulas over predicates);
- the **preconditions** of an action is set of predicates (or a free-form logical formula) that must be true in the current state for an action to be applicable (executable) in that state;
- the **effects** of an action are a set of predicates that will be added or deleted from the current state, creating a new state;
- a **domain** describes the key elements of a planning domain, comprising the set of valid predicates (properties of the world) and the actions (transition function) available in the domain;
- a **problem** is a specific instance of the domain to be solved, with a set of objects, an initial state and goal state that must be reached; and
- a **solution or plan** is a finite sequence of operators available to the domain that when applied to the initial state satisfy the goal state [22].

Some problems may have no solution (i.e. there is no plan that can transform the initial state into the goal state), while some problems may have multiple solutions. A plan is said to be optimal if it is the shortest plan that achieves the goal. To take advantage of the possibility of multiple plans for a given problem, some planners relax optimality constraints to speed-up search and find a suboptimal plan. Note that the sequence of actions may be represented in a tree-like structure to better describe dependency and order between actions, although the linear idea of sequence is used to simplify the relationship of preconditions and effects.

A. Planning Languages

The formal elements we describe in this section are used in the definition of planning languages, the first of which was implemented for *Stanford Research Institute Problem Solver* (STRIPS) system. STRIPS is a planning system created in 1971 [8] and became important due to its formalization of the description of the world, providing much of the structure for planning problem specifications we describe above. In order to specify more complex domains, later planners defined a series of extensions and planning languages. These languages were consolidated into the *Planning Domain Definition Language* (PDDL)[14], created in 1998 to be the standard language for AI planning.

B. Classical Planning Algorithms

Different algorithms solve planning problems using different approaches on how to deal with the combinations of action sequences that yield a solution. The most straightforward planning mechanism consists of a forward search in the state-space, checking which actions can be applied to the current state generating further states until the goal state is found. For much of the evolution of planning algorithms, forward search did not yield efficient results, due to the need for very good heuristics to avoid the large branching factor inherent to this type of planning. However, later research has shown that such heuristics are possible for very efficient planning [9]. In this paper, we use a heuristic search-based planning algorithm.

V. BioPlan: A classical planning extension to BioCrowds

BioCrowds is both a model (see Section III-A) and simulator for interacting agents based on fixed markers in space. Agents move through the environment trying to reach a sequence of 'waypoints', which must be manually adjusted by a human designer. In order to reach their destinations, agents move by trying to occupy free markers around them, without differentiating whether the lack of free adjacent markers is due to walls or the transient occupation of a marker by another agent. As each agent tries to occupy the markers in the direction of the next goal position, a problem arises when there are not enough intermediary destinations to guide them, which, as a result, become stuck not knowing whether to try alternative paths or to wait for a nearby marker to become free. Planning becomes an attractive approach at this point, to find a path between the current position and one of many goal positions defined as declarative goals. The use of declarative goals frees a designer to simply specify what the goals are (be they positions in the map, or other, abstract states), and let the planning algorithm generate the intermediary positions in the map, as well as the positions that are to be avoided to prevent agents getting stuck. Nevertheless, classical planning not only costs too much processing power for a group of agents, but also removes liberty as the plan gives a description of each step. However, if we describe a goal state and how to perform actions to reach such state it is possible to remove some intermediary movement actions to add liberty. Moving from a desired point to a desired state does not mean we want to occupy a position, but achieve several properties of the environment.

In order to plan reasonably fast, agents are grouped based on similar initial and desired states. The group share the same map with the same action-points, specific places where agents can achieve certain abstract features. Each group may consist of one or more agents. Since multiple agents share the same initial and goal configurations we can plan once for each group. Agent groups have a set of shared attributes for path planning that are inherited by all agents in a group, but some attributes are randomized per agent, e.g. speed. Such different path planning attributes naturally lead to the actual paths taken by the agent being different even for agents in the same group, as speed and collision may create a different local scenario for each agent. Although there are many more attributes, we summarize in Table I the attributes that are relevant for our model. The number of agents each group have as an integer. The source and target positions of the route are required by the simulator, the source position is where the agents are spawned and the target position where each instance is destroyed. The positions are represented by nodes, a discrete grid representation of the continuous environment. The simulation stops when all agents reach their target. Path mode tells how the route must be considered, as only two points, a path planning problem or a classical planning problem. Freedom represents how many intermediary positions we want to remove from the route after planning is done, without those points our agents can avoid other agents by going a little off the track, instead of waiting the crowd to give some space to get near the original route. The start and goal states of the group must be represented, like all agents starting without resources and the goal is to have them. Speed, proxemics and color are simulator variables that are randomly generated values within constraints before execution, therefore there is a minimum and maximum speed, different levels of proxemics and colors available for the agents.

TABLE I. Group attributes

Attribute	Possible values	Description
agents	1 to ∞	Number of agents from group
route	Array of nodes	Source and target destination
path mode	direct, A*, classical planning	Path generation method
freedom	1 to ∞	Number of edges between nodes of path
start	Array of requirements	Initial state of a group
goal	Array of requirements	Desire state of a group
speed	Float	Agent maximum speed without obstacles
proxemics	Float	How close a marker must be to be used by the agent
color	RGB	The color used by agents, path and marker's connector

The next important element in a simulation after the groups were described is the map. The map is important for both planning and simulation, as obstacles define the environment and action-points define the different ways agents may achieve a certain feature. The map is considered static and described in a discrete form to the planner, making the goal of planning the change of the agents position, internal state and resources instead of the world itself for most domains. Changing the map is usually a good idea for coordinated groups, as agents plan to act at specific places to reach the same goal, but becomes a problem for our crowd model. Imagine the goal of a group to be that of moving behind a fireproof door during an evacuation.

Here, after the first agent to reach the door must open the door, everyone enters the safe room and the door is then closed by one them. This is a coordinated action. The problem can be broken into smaller parts (such as open, enter, close) each of which can be executed either by a single agent, or by subgroups of agents.

If implemented by our planner all agents would have to close the door behind them as no agent knows if they are the last one to reach the door. This exemplifies very well the crowd planning problem, the agents do not recognize being part of a group and do what they need to satisfy their needs, even if all agents have the same goals. The second problem is the single entity approach, in order to make the last agent close the door, each agent would need to plan individually and one would need to be the designated as the agent responsible for closing the door. Thus, we concentrate on a static map as the simulator cannot display mutable features of the environment, such as doors being open or closed.

Our model is based on planning paths for all agent groups before the simulation starts. Instead of planning for the N groups at the same time, making the problem more complex for a large N, our model breaks the problem down for each group, making the simulation not only simpler but also more realistic. The simulation is more realistic because in the real world, agents follow a path and react to other individuals as obstacles that alter their path locally. Thus, locally, the behavior of the agent is more important than the plan. Our approach is also simpler, because taking into account all groups at the same time would yield a plan that represents the position of the groups in a discretized and deterministic way, which is not true at simulation time as collision avoidance and speed differences make the agents diverge locally from the optimal path.

The desired state can involve several propositions beyond the position, with several being dependent and possible to achieve in different ways according to the initial state and actions available. Unlike path-planning, an agent using our planner may desire to explicitly avoid a certain property, such as starting *dirty* and desiring to be *clean* (not *dirty*).

As the simulation is executed, the agents move at different speeds (that are not represented by the planner) based on random distribution of the markers and individual speed settings. Consequently, blindly following the plan to avoid other groups at the same time is not enough because of those non-deterministic attributes. If we used a planner in which those details were explicitly represented, the resulting plans would eliminate the realism of the situations created by such randomness, as agents would re-plan only if a large number of agents completely blocked their passage to their destination. Replanning would also create a processing bottleneck as only individuals require replanning based on their current situation. Therefore the simulator received few modifications to accept input from our planner.

Adding the output of a classical planner as input for the simulator seamlessly requires some constraints on the output of the planner so that it is interpreted correctly by the simulator. Since the simulator expects a path and not a plan as input, a conversion is required. The planner requires the map of the environment as well as the simulator and a conversion process was required to automate the generation of group attributes,

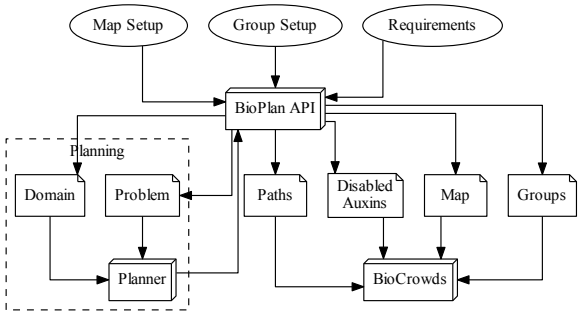

Fig. 2. Flow of the proposed model

adjacency information of the map, how actions interact with the world-state and how agents execute the plan or deal with a failure in planning. In other words, it requires a single input to describe both planner and simulator inputs.

VI. Implementation

In our implementation, we use a forward-search classical planner in such a way as to focus on a single group at a time, instead of generating several domain and problem inputs for each group, certain specific points of the planner were modified. Instead of planning for every group at the same time, the system focuses on each group leading to the more plans being generated, though each individual plan has fewer steps and a smaller state space. Otherwise each group would be an object available to instance more actions, which would result in a very large number of operators, leading to a large branching factor.

Search-based planners have to deal with problems with large branching factors due to the large number of possible concrete operators that must be tested for each new state that appear during the search process. Moreover, once concrete operators are generated and the planner knows the exact size of the state representation, we generate an internal binary-vector based representation that allows for very efficient planning both in terms of runtime and memory usage.

We integrate all of these systems in a processing pipeline, illustrated in Fig. 2. Each subsystem (the classical planner and the simulator) requires its data to be converted before they can communicate with each other. To address that, we implemented the BioPlan API, which handles all convertion processes. The designer must specify map, groups and transition rules to be used by the planner, as shown in Listing 3. Different groups using different maps generate more map and disabled auxins files, and since the BioCrowds simulator expects only one of each the designer must choose the actual simulator input to be used. Actions for moving throughout the space are statically defined and reused in all simulation domains, and additional rules for specific simulation requirements are converted into additional planning operators. These rules are then used to create valid PDDL actions. An example problem used throughout this paper is shown Listing 1, with the corresponding domain shown in Listing 2. The problem corresponds to one specific scenario, in this case, the objects are agents,

position nodes and the different requirements involved, the initial state with the current configuration and the goal state we desire to achieve. The transitions are defined in the domain as follows: two actions to move the agents, one for orthogonal and another for diagonal movement; and an action that executes a certain task if all requirements have been fulfilled and the agent is at one of the possible places where that task can be performed. Here, the planner gives priority to the orthogonal movement because these actions are defined first in the domain description, therefore the orthogonal movements are expanded first and yield a shorter path even for solutions with the same number of actions.

```
(define (problem pb1)
  (:domain crowd)
  (:requirements :strips :negative-preconditions)
  (:objects
    a0 a1 a2 n1 ... n964
    null use_bathroom get_papers)
  (:init
    (requirement n43  use_bathroom null)
    (requirement n963 use_bathroom null)
    (requirement n632 get_papers   null)
    (agent a0) (agent a1) (agent a2)
    (at a0 n478)
    (at a1 n897)
    (at a2 n455)
    (have a0 null)
    (have a1 null)
    (have a2 null)
    (adjacent_1 n43 n44)
    (adjacent_1 n43 n85)
    (adjacent_2 n43 n86)
    (adjacent_1 n44 n43)
    (adjacent_1 n44 n45)
    (adjacent_2 n44 n85)
    (adjacent_1 n44 n86)
    ...
    (adjacent_1 n964 n922)
    (adjacent_1 n964 n963))
  (:goal (and
    (at a0 n777)
    (at a1 n901)
    (at a2 n498)
    (have a0 use_bathroom)
    (have a1 use_bathroom)
    (have a2 use_bathroom)
    (have a2 get_papers))))
```

Listing 1. PDDL problem example

Groups are declared with the number of agents, route (start/goal positions), path mode and when planning is used, a degree of freedom and a set of requirements to fulfill along the path. We have defined three path following modes that can be used depending on the simulation needs of the designer: direct mode, A* mode, and planning mode. The direct path mode is the one without any intermediary points to be reached, there is only the initial and goal checkpoints, in Fig. 3 they appear as the blue points with a square around, with the walls in black and the avoidance nodes in gray. This mode gives too much freedom of movement to an agent, as the goal may be too far or even beyond several walls in a complex scenario. BioCrowds agents do not bounce or walk back, they always move trying to minimize the distance between themselves and the next goal point, so a simple corridor is enough to make them hit a wall and stay there until the end of the simulation. The direct mode should only be used to check interactions between crowds in

```
(define (domain crowd)
  (:requirements :strips :negative-preconditions)
  (:predicates
    (agent ?agent)
    (adjacent_1 ?source ?destination)
    (adjacent_2 ?source ?destination)
    (at ?agent ?source)
    (have ?agent ?goal))
  (:action move_1
  :parameters (?agent ?source ?destination)
  :precondition (and
    (agent ?agent)
    (adjacent_1 ?source ?destination)
    (at ?agent ?source))
  :effect (and
    (not (at ?agent ?source))
    (at ?agent ?destination)))
  (:action move_2
  :parameters (?agent ?source ?destination)
  :precondition (and
    (agent ?agent)
    (adjacent_2 ?source ?destination)
    (at ?agent ?source))
  :effect (and
    (not (at ?agent ?source))
    (at ?agent ?destination)))
  (:action do
  :parameters (?agent ?source ?previous ?goal)
  :precondition (and
    (agent ?agent)
    (requirement ?source ?goal ?previous)
    (at ?agent ?source)
    (have ?agent ?previous)
    (not (have ?agent ?goal)))
  :effect (and (have ?agent ?goal))))
```

Listing 2. PDDL domain example

open scenarios, which is one of the goals of the simulator and explains the reason for this mode to be available. On the other hand, the A* mode restricts the agents to strictly follow a path instead of directly trying to reach a target position as in the direct mode, the path is shown as a sequence of blue points without a square in Fig. 4. The only leeway an agent following A* mode has regards the proximity allowed for an agent to be considered in range of the current checkpoint, which only amounts to a perception mechanism to avoid cluttering agents into a single point at the same time. Classical planning mode is not much different from the A* mode, as both search for a sequence that fulfills an agent's goals. The difference is what the sequence contains, whereas A* is only concerned with a sequence of points, classical planning mode is concerned with a sequence of actions that move an agent from point A to point B in this case. Classical planning can also deal with actions beyond movement, but the other actions are important only to the planner as no animation related to those actions is supported in the current version of the BioCrowds simulator. A* is already implemented in the simulator and we prefer to let it return a full path as expected while the path our classical planner generates could be relaxed within a certain degree of freedom. This degree of freedom can be used to avoid an unnatural movement by the agents, removing intermediary movement-only actions among the plan and letting the agent to choose a path during execution. Examples of different degrees of freedom can be seen at Figures 5 and 6.

The degrees of freedom defined by the user can also be

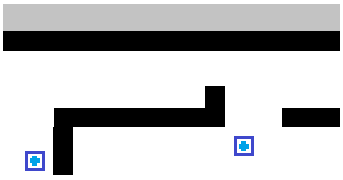

Fig. 3. Goal and target of direct mode are the only points given to the simulator using direct mode

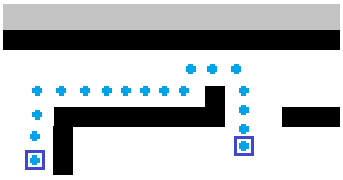

Fig. 4. Path of A* mode gives a sequence of points to act as intermediary targets to help the agent reach the final target

explored by an automatic mechanism that identifies straight lines within the movement actions of the plan and removes the intermediary movement-actions without removing the points that define curves, an example of such removal for an original path in Fig. 4 is the path of Fig. 7. Since the points are always distributed in the grid and Moore neighborhood is used to define adjacent cells there are 8 cases of straight lines, as shown in Fig. 8, which makes it simple to recognize one of the possible straight lines and define our waypoints. Fig. 9 shows the algorithm that eliminates intermediary waypoints along straight lines using Moore neighborhood.

Before we start the search for a path we need to describe the groups and map for the planner, as shown in the first steps of Fig. 10. We start with the map to extract information about which nodes can be accessed. There are three types of nodes so far: clear nodes that can be accessed, wall nodes that are always closed and avoidance nodes that are always considered closed for planning but clear for the simulator. This information is used to avoid creating markers at specific regions in the simulator. Adjacency between nodes in the maps are converted to predicates describing edges to be used by the

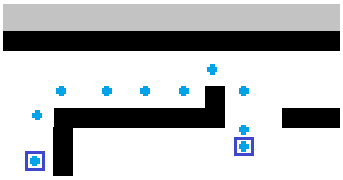

Fig. 5. Planning mode with freedom 2 gives a path with less intermediary targets

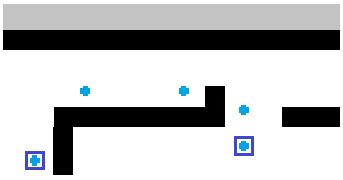

Fig. 6. Planning mode with freedom 5 gives a path without sufficient points, which may lead the agents to a wall which they behavior is not enough to avoid.

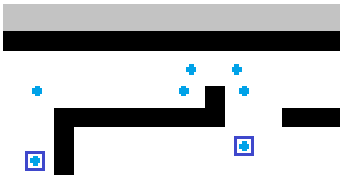

Fig. 7. Planning mode with automatic waypoint is a solution to find a small set of points enough for the agents to reach their goal.

	NW	N	NE	
	W	C	E	
	SW	S	SE	

Fig. 8. Moore neighborhood as the 8 cells around a central cell.

planner. The requirements and a subset of the group attributes are part of the problem as initial or goal state while the domain is a static file with the movement and a generic action that can be performed based on the requirements previously defined. Finally, we executed planner for each group defining planning as operating mode, and, if successful we modify the plan according to the freedom of the group in question.

A. *Domain Knowledge*

Domain knowledge is a series of optimizations about a specific domain that can yield a better or faster solution when used. Several problems have to deal with the fact that

```
1:  procedure FIND_WAYPOINTS(path)
2:      waypts ← [first element of path]
3:      remove first element of path
4:      len ← length of path
5:      i ← 0
6:      while i < len do
7:          if path[i] = action position
8:              waypts push path[i - 1]
9:              i ← i + 1
10:             continue
11:         for dir in [N,S,E,W,NW,NE,SW,SE] do
12:             last ← last element of waypts
13:             if last at dir of path[i]
14:                 last ← path[i]
15:                 i ← i + 1
16:                 while last at dir of path[i] and i < len do
17:                     last ← path[i]
18:                     i ← i + 1
19:                 waypts push path[i - 1]
20:                 break
21:     return waypts
```

Fig. 9. Find lines among the points and remove the intermediary points, keeping only the points where the agent must turn or execute a non-movement action

```
require 'BioPlan'
group_1 = {
  :agents => 15, :route => [839, 666],
  :mode => BioPlan::PLANNING,
  :freedom => 3,
  :start => ['null'],
  :goal => ['receive_food']
}
group_2 = {
  :agents => 7, :route => [839, 540],
  :mode => Bioplan::PLANNING,
  :start => ['null','use_bathroom'],
  :goal => ['receive_food']
}
group_3 = {
  :agents => 5, :route => [839, 540],
  :mode => BioPlan::ASTAR
}
groups_1 = [group_1, group_2]
groups_2 = [group_3]
requirements = [
  {
   :at => 968, :require => 'null',
   :able => 'use_bathroom'
  },{
   :at => 440, :require => 'use_bathroom',
   :able => 'pay_food'
  },{
   :at => 165, :require => 'pay_food',
   :able => 'receive_food'
  }
]
map_1 = Image.load_bmp('map1.bmp')
BioPlan.setup(groups_1, map_1, requirements)
map_2 = Image.load_bmp('map2.bmp')
BioPlan.setup(groups_2, map_2, requirements)
```

Listing 3. API example

```
1: procedure BIOPLAN::SETUP(groups, map, required)
2:     clean map
3:     mark disabled map nodes
4:     save marked nodes to file
5:     add required to initial state
6:     for g in groups do
7:         add current state of g to initial state
8:         add goals of g to goal state
9:     extract graph from map
10:    add adjacencies to initial state
11:    save map to file
12:    save problem to file
13:    for g in groups do
14:        if mode g is PLANNING
15:            plan ← planner(domain, problem, g)
16:            if plan found
17:                path ← []
18:                for action in plan with index i do
19:                    if action=move and i%g.freedom=0
20:                        path push position
21:                    else if action=do
22:                        path push position
23:                if only waypoints
24:                    path ← Find_waypoints(path)
25:                save path to file
26:            else
27:                print warning for g
28:                Downgrade g to path planning
29:    save groups to file
```

Fig. 10. With the map loaded we generate the required paths for each group and save to files to be used as input in the simulator.

Fig. 11. Regions to avoid during planning are marked in gray and walls in black.

the domain knowledge is incomplete or non-existent. The map information containing only walls and obstacles as well as starting and goal points is not enough to generate an acceptable plan for the simulation. Most problems faced by the BioCrowds model come from collision avoidance from narrow paths and agents being thrown away from their route by other agents. Adding more input to the map can yield a better path. Some of those problems are explained in this section with more detail.

1) Map Contour: Without a predefined path, agents tend to use a movement pattern similar to a best-first search, thus approaching the goal position without taking into consideration the contour of the map. This usually results in failure (agents getting stuck) if the distance is too long, as more walls may appear in the way. The failure persists even with a predefined path, with agents becoming trapped by an irregular wall. We add further knowledge of avoidance nodes to be avoided in the map to avoid passing near those walls and have agents becoming stuck in a local minimum. This knowledge is used by the planner to both speed-up (eliminate nodes from the search) and add designer control, leaving such nodes as clear nodes during simulation and letting agents use them as a last resource when more individual space is needed. Several tricks can be used, such as: aiming to pass through a door targeting exactly its midpoint; avoiding walking too close to walls in passages to avoid being surprised by other agents; and avoiding traversing queues of waiting agents, but rather pass behind the queues. Some of those examples can be seen in Fig. 11 in which gray squares represent the regions to avoid and black squares represent the walls.

The map is defined by nodes, nodes can represent three types of terrain: clear, walls and avoidance. Clear nodes are always considered open, while walls are always considered closed. Avoidance nodes are considered closed during planning, but are open during execution. Avoidance nodes are useful not only to remove certain paths from consideration, but also to provide more control over the resulting paths, something any designer wants. Constraining where desired and letting the group free otherwise. The map walls represent places where no auxin exists, creating a lack of possible paths for agents. The terrain can be seen at Fig. 18, with a zoom in the upper left corner, where green squares are the nodes and the black dots the auxins. Connected nodes are possible paths and auxins represent free space that agents use

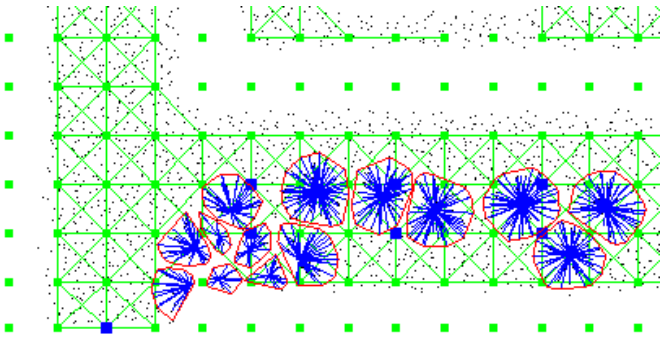

Fig. 12. Bathroom bottleneck.

while moving to conquer more individual space in their goal direction. The avoidance system was inspired by the preference system of [23], but instead of being part of the behavior it became part of the planning process.

2) Perception and choice: One of the main problems is how much liberty the agents can have. The idea of group here would be broken if some agents took a different path. In some situations this makes sense, like in a bathroom choice according to the size of the waiting line, but the problem persists not on how the agents would perceive this, but in how effective it is to leave them free to explore based solely on perceptions or decision points within a plan. This can be seen in the bathroom situation, Fig. 12, as the agents reach the bathroom they start coming back and compete for their previous space, breaking the lane formation created by the BioCrowds model. Some agents try to go to the free side while others make a poorer choice and end up stuck until most agents find their way to the bathroom. One of the main problems of removing intermediary points is that they are not enough to eliminate the bottleneck regions that happen when agents achieve one of their goals and suddenly start coming back.

Instead of using a reasoning system during run time, the idea is to let each agent have more liberty without abandoning the path altogether, removing some intermediary points. The simple removal of intermediary points can lead to failure in some cases and, consequently, identification of such cases is important to understand how free the agent can be while staying close to the path in narrow passages.

3) Heuristic: Most problems can be solved faster using a relaxation strategy to focus on the relevant attributes. Working with this subset of the actual problem may lead to a sub-optimal solution with less resources used. Finding the heuristic to a specific domain is a problem by itself and since a classical planner have no idea which domains may be presented as input a domain dependent heuristic is not part of the planner. Methods to relax the problem definition are used. One of the most basic general heuristics is the Hamming distance [24], since the problem is completely observable it is possible to compare the current state with the desired state and count how many propositions (state variables) are different. Most propositions are modified by a single action, but several properties of the world may be affected by the same action, making this heuristic inadmissible for several problems and yielding suboptimal plans. Problems without much information about the desired state also suffer with the heuristic, as for most states the heuristic function returns the same value while slowing the search down to compute the difference between states, effectively searching using a breadth-first strategy. In order to this comparison fast we enumerate all propositions that may appear during the planning process and give an index to each. Using a vector of bits we consider each bit as a proposition based on their index and the value as being true or false in the current state.

VII. Results

We created three scenarios to test our approach: one with several randomly placed walls and two more realistic ones, with scenarios based on actual locations. The path modes already supported by the BioCrowds (direct and A*) simulator were exposed to the API. In every scenario the direct mode shows the problem of not having an entire path to follow, with some agents becoming stuck in the first perpendicular wall they encounter, while others wait the agents in front of them to move. Using the internal A* generated path is still a good choice when the group only wants to reach some point, but kills the entire liberty. The avoidance system, removing node connections without removing auxins, fits the A*, as both planners only use nodes. When there is a set of requirements to fulfill, the classical planner is the preferred tool, making all the choices before the simulation starts and creating a robust path for the group that fulfill the requirements along the way. Since both A* and planning can fail, and failure at simulation time is undesirable, we need to find a path before the simulation starts. If A* fails there is no path from the source point to the goal point and reaching or not the goal is irrelevant to the simulation. However, since the agents must occupy space on the map during simulation, we modify the group path to direct mode, so that the agents will keep trying to achieve their goal without success while their presence in the map can still affect other groups.

If planning fails, the requirements may be unobtainable and the designer may give-up and opt to switch for only path planning with A*. Other reasons for failure include avoidance nodes, which can block some paths if not used carefully by the designer. In such cases, a designer may need to manually remove avoidance nodes to achieve the intended simulation. Note that failures to reach a destination due to avoidance nodes may affect even paths generation via the A* mode, there is no guarantee that giving up requirements will be enough to find a path if the avoidance nodes still block the target position of the group. This way, no group has to be destroyed because of path/requirements failure and the simulation will show what was predicted, total failure (agent cannot reach destination with requirements) or partial failure (agent reach destination without fulfilling requirements) by dropping additional requirements through the removal of some desired propositions. The main problem with the automatic relaxation of a map by removing avoidance nodes when the planner fails, is how a single group affects the map used as input by the simulator, and how to enforce avoidance of some parts of the map to simulate a real event. Here, the planner is not aware of the cause of failure, it simply infers that there is no sequence of actions that can be executed to reach the desired state. The cause may be the map, therefore relaxation of map preferences may help, otherwise the requirements need access to completely

disconnected regions of the map and are impossible for the agents to reach. In such cases only a modification of the requirements would be enough to successfully find a plan, therefore we modify the goals of the agents, simulating agents giving up on tasks they do not know how to solve. Real events being recreated may use more than avoidance nodes to improve path quality, they may remove a path that people consider too dangerous or that is not perceived at the time as a viable path. There are two ways in which a classical planner can deal with this problem without adding too much complexity: recreating the problem with fewer nodes and having several levels of avoidance nodes to remove as planning failed; or letting the designer know that the planner is failing and decide what to do with the current map. We chose the designer as the first option to deal with this problem, since creating different levels of avoidance is more expressive but can be time-consuming, as only the designer can say which tasks or nodes should be refined first.

The maps we use for experimentation have 42x24 nodes and can yield 5000 propositions of node adjacency using a Moore neighborhood (8 connected cells around), this step is generated from a char-based map (an image can also be used, conversion is supported by the API). Although the maps used are grid-based, the idea of connectivity based on Cartesian coordinates is a constraint for different connectivity systems (like hexagonal tiles) and nodes are referenced by an identifier instead. A new map input method would be required to generate the connections for the planner and few modifications in the simulator to actually display the new map system correctly.

A. Random Scenario

We created the random scenario as the first environment to illustrate the output of the tool. Fig. 13 shows the scenario map, while Fig. 14 shows the two groups and the connectivity between the nodes in the map. Here, the dark blue markers represent the control group (without a planned path, i.e. in direct mode) and the clear blue markers represent the group that uses path planning mode (the path planned group). The path planned group must travel from left to right, while the control group must go from right to left. The path planned group must avoid a long line to reach its goal, achieve part of its goals with an action in a specific point of the map and reach the other side. The control group just tries to reach the other side, but walls with varying angles make the goal much harder than simply crossing the map. The control group was used to understand how much competition exists between the agents and was the motivator for the use of avoidance nodes, as many agents get stuck in this scenario and how to enforce planning to avoid such dangerous regions.

B. Lab Scenario

In the Lab scenario, Figures 15 and 16, the narrow paths become a problem that requires more than what the avoidance nodes may help, as other agents push others inside near rooms and the chance of groups collision being extremely high in the narrow corridors. Perhaps the problem is the perception of agents or the design of the space, a more accurate map is required to see if the environment is working against the agents. This environment have two bathrooms available and one printer room, two groups want to use the bathroom and go back to their respective rooms while one group wants to get the papers they printed and also use the bathroom. It is possible to see that each group tries to reach the nearest bathroom and the paper group goes around the lab in order to achieve both goals.

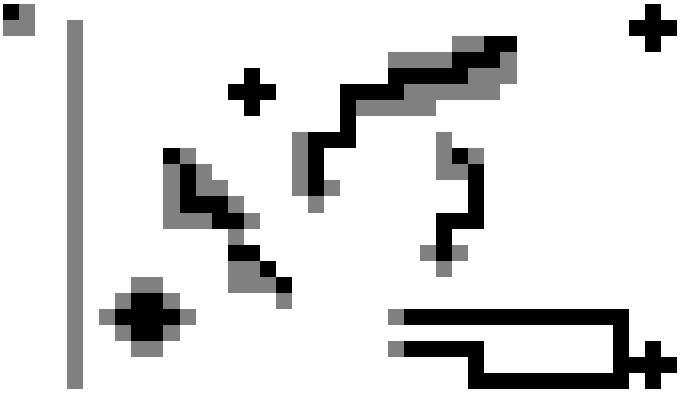

Fig. 13. Random map input as image file

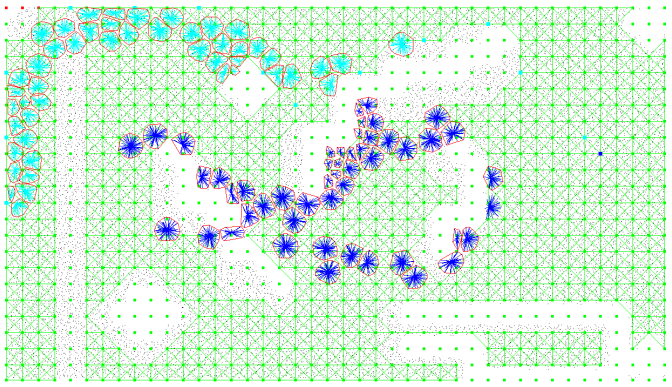

Fig. 14. Random scenario

C. Snack Bar Scenario

In the snack bar scenario, Figures 17 and 18, the agents can go to a bathroom, pay for the food at the first counter and receive the food at the second counter. Some requirements were created to this scenario: each agent can buy only if they

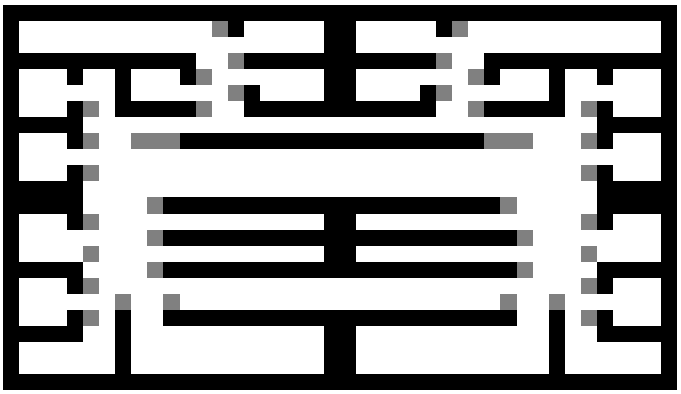

Fig. 15. Lab map input as image file

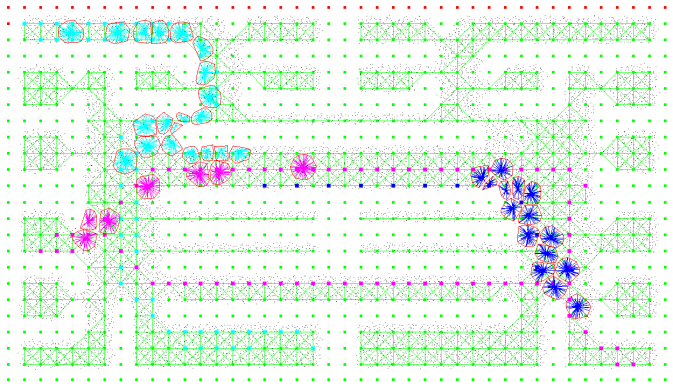

Fig. 16. Lab scenario

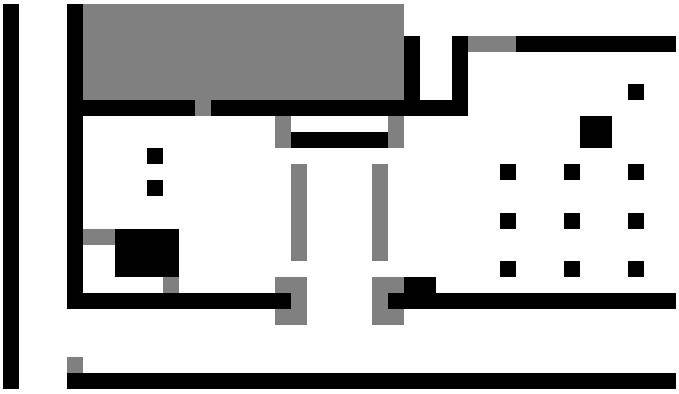

Fig. 17. Snack bar map input as image file

do not need to go to the bathroom first, once they bought they can receive the food. Some groups already start with no bathroom need while others only want to buy the food to later eat. The liberty here is that the bathroom can be easily reallocated and more bathrooms can exist, but the goal was to simplify the entire process. Therefore there is no need to define the map and where the actions are possible at the same time, the map only holds the walls and the requirements can be obtained at specific nodes once their preconditions are satisfied.

Some scenarios may limit the possible paths an agent could take, due to knowledge about the environment (agent does not know about a path during path planning) or restrictions imposed by the class of the agent, like a kid being able to pass where an adult cannot and an employee being able to enter any room in a store, while the customers are restricted to the main room. In order to better define such map restrictions we need to describe and relate each map to a group of agents. Here, we use the avoidance system (Section VI-A1) to eliminate map positions from the planning process and create different restrictions to each group. Since each group can already plan independently (using the avoidance system) there is no need to add any control in the planning, just use the respective map for the current group. To evaluate the different restrictions on different groups, we extend the snack bar scenario by adding employees that can take advantage of the backdoor and can access the other side of the counter. In order for this to work we use the map of Fig. 19 to be the new default map and use the map of Fig. 18 as the custom one used by the customers.

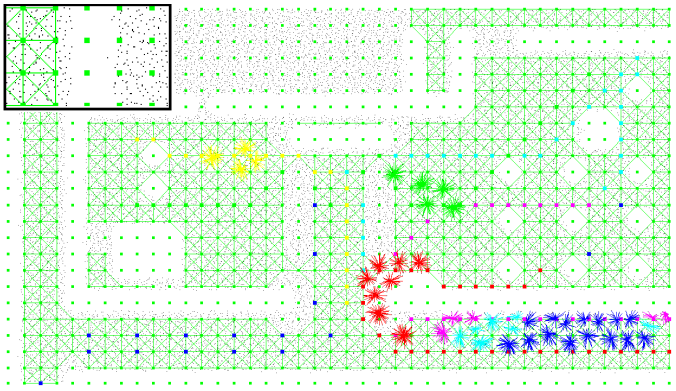

Fig. 18. Snack bar scenario

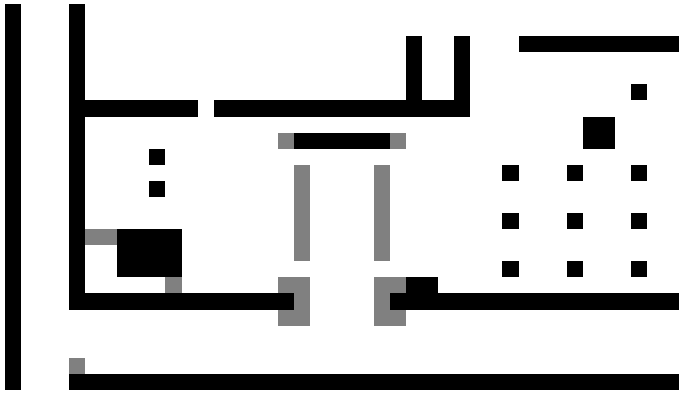

Fig. 19. Snack bar scenario without avoidance nodes for the employees

VIII. Conclusion and Future Work

In this paper, we have described a novel approach based on classical planning for the generation of group behavior in crowd simulation. Classical planning and some domain knowledge can be applied to yield custom paths to BioCrowds, adding not just complexity to the paths but lowering the chance of failure (in the form of agents being unrealistically stuck in the map) as well as providing a more meaningful way to represent subgoals for the agents being simulated. Although our current implementation shows a relatively small set of subgoals being expressed by each agent group, our approach can be easily employed to help develop more realistic scenarios without the need to explicitly define alternative paths to subgoals, ultimately allowing a simulation designer to focus on higher-level agent behavior. Although one may argue that the centralized and deterministic solution of the planner undermines the approach of crowd simulation with distributed behavior, two considerations can be used in favor of our implementation. The first is that knowledge of the markers is exactly the same for all BioCrowds agents. Every agent perceives the world in the same way during simulation, and no space is invaded because of different point of views, consequently centralized marker knowledge is already assumed by the BioCrowds model. The second is that the generated plans are not followed blindly by the agents, as each agent

tries to solve local problems based on their specific situation during simulation. Thus, our plans are merely guidelines for intermediary points that agents follow as they try to reach their goal. Agents are grouped just for time-saving purposes as all agents would plan for the same goal from the same starting point. The only limits of planning being time and memory for the process, a thousand agents would plan the same way as a single agent, the difference would happen just in the simulation as fewer markers would be available per agent.

Our implementation was validated through a number of scenarios, however, our current approach has a number of limitations. In maps with narrow choke points, our approach tends to require designer intervention through explicit specification of avoidance points. Although our focus so far has been on the integration of a planner into the crowd simulation pipeline, we aim to expand our work in a number of ways. First, we aim to investigate other mechanisms for node avoidance besides designer-expressed avoidance nodes, and instead generate them automatically based on generic sets of rules (e.g. using cellular automata). Second, we aim to merge different preferences to be used for different subgroups within the same map, e.g. to avoid certain map regions as only authorized personnel may enter, in the form of nodes instead of using different maps for each group.

ACKNOWLEDGMENT

We thank Soraia Müsse, Vinícius Cassol and Cliceres Mack Dal Bianco for the discussions that led to the writing of this paper, and for access to the source code of the BioCrowds simulator for our experiments.

REFERENCES

[1] J. Orkin, "Three states and a plan: the AI of FEAR," in *Game Developers Conference*, vol. 2006, 2006, p. 4.

[2] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha, "Real-time navigation of independent agents using adaptive roadmaps," in *Proc. 2007 ACM symposium on Virtual reality software and technology*. ACM, 2007, pp. 99–106.

[3] A. Treuille, S. Cooper, and Z. Popović, "Continuum crowds," in *ACM Transactions on Graphics*, vol. 25, no. 3. ACM, 2006, pp. 1160–1168.

[4] T.-Y. Li, Y.-J. Jeng, and S.-I. Chang, "Simulating virtual human crowds with a leader-follower model," in *Computer Animation, 2001. The 14th Conf. on Computer Animation. Proceedings*. IEEE, 2001, pp. 93–102.

[5] J. Funge, X. Tu, and D. Terzopoulos, "Cognitive modeling: knowledge, reasoning and planning for intelligent characters," in *Proc. 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1999, pp. 29–38.

[6] M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah, "Declarative & Procedural Goals in Intelligent Agent Systems," in *Procs. 8th Int. Conf. on Principles and Knowledge Representation and Reasoning*, D. Fensel, F. Giunchiglia, D. L. McGuinness, and M.-A. Williams, Eds. Morgan Kaufmann, 2002, pp. 470–481.

[7] A. de Lima Bicho, "Da modelagem de plantas à dinâmica de multidões: um modelo de animação comportamental bio-inspirado," Ph.D. dissertation, Universidade Estadual de Campinas, 2009.

[8] R. E. Fikes and N. J. Nilsson, "STRIPS: A new approach to the application of theorem proving to problem solving," *Artificial intelligence*, vol. 2, no. 3, pp. 189–208, 1971.

[9] B. Bonet and H. Geffner, "Planning as heuristic search," *Artificial Intelligence*, vol. 129, no. 1, pp. 5–33, 2001.

[10] M. C. Magnaguagno and F. Meneguzzi, "BioPlan: An API for classical planning on biocrowds," in *Computer Games and Digital Entertainment (SBGAMES), 2014 Brazilian Symposium on*. IEEE, 2014, pp. 11–20.

[11] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," *ACM SIGGRAPH*, vol. 21, no. 4, pp. 25–34, 1987.

[12] L. M. Flach, V. J. Cassol, F. P. Marson, and S. R. Musse, "A procedural approach to simulate virtual agents behaviors in indoor environments," in *Intelligent Virtual Agents*. Springer, 2013, p. 448.

[13] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory and Practice*. Elsevier, 2004.

[14] D. McDermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins, "PDDL-the planning domain definition language," 1998.

[15] A. Coles, A. Coles, A. G. Olaya, S. Jiménez, C. L. López, S. Sanner, and S. Yoon, "A survey of the seventh international planning competition," *AI Magazine*, vol. 33, no. 1, pp. 83–88, 2012.

[16] F. Meneguzzi and L. De Silva, "Planning in BDI agents: a survey of the integration of planning algorithms and agent reasoning," *The Knowledge Engineering Review*, vol. FirstView, pp. 1–44, 9 2013.

[17] S. R. Musse and D. Thalmann, *A model of human crowd behavior: Group inter-relationship and collision detection analysis*. Springer, 1997.

[18] N. Pelechano, K. O'Brien, B. Silverman, and N. Badler, "Crowd simulation incorporating agent psychological models, roles and communication," DTIC Document, Tech. Rep., 2005.

[19] M. Sung, M. Gleicher, and S. Chenney, "Scalable behaviors for crowd simulation," in *Computer Graphics Forum*, vol. 23, no. 3. Wiley Online Library, 2004, pp. 519–528.

[20] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[21] B. Solmaz, B. E. Moore, and M. Shah, "Identifying behaviors in crowd scenes using stability analysis for dynamical systems," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 34, no. 10, pp. 2064–2070, 2012.

[22] B. Nebel, "On the compilability and expressive power of propositional planning formalisms," *Journal of Artificial Intelligence Research*, vol. 12, pp. 271–315, 2000.

[23] V. J. Cassol, F. P. Marson, M. Vendramini, M. Paravisi, A. Bicho, C. Jung, and S. Musse, "Simulation of autonomous agents using terrain reasoning," in *Proc. 12th IASTED Int. Conf. on Computer Graphics and Imaging, Innsbruck, Austria. IASTED/ACTA Press*, 2011.

[24] R. W. Hamming, "Error detecting and error correcting codes," *Bell System technical journal*, vol. 29, no. 2, pp. 147–160, 1950.