Distance-Based Construction Behaviors for Dynamic Grid-Based Worlds

Charles E. Myers*, **Pushpak Karnick**†
DigiPen Institute of Technology
9931 Willows Rd NE
Redmond, WA 98052

Abstract

This paper proposes a method for the development of AI for autonomous agents in game worlds modeled by fixed regular grids. This approach uses context behaviors driven by distance maps to support multiple agents in a dynamic environment. This paper introduces the notion of hierarchical distance maps which allow for higher-level goals to be easily specified by designers. We also discuss potential applications of our approach in the design and development of agents and behaviors in the block world genre.

Introduction

Grid-based worlds have recently seen a resurgence with the rise of Dwarf Fortress, Minecraft and other, similar games (Lynley 2013). These games represent a new genre, referred to as *block-world* due to being stored in fixed regular grids, and have been heralded as combinations of deep and free-form gameplay. Unlike the majority of games with concrete, developer-specified goals, block-world games instead tend to provide the player with a large, dynamically generated environment, a technology tree and freedom (Duncan 2011).

While there are many elements which constitute gameplay in a block-world game we will focus on two particular aspects - resource collection and construction. It is important to note that while survival is one of the most common goals in computer games, in the case of the block-world genre it currently serves as the primary motivator only for early decisions. As players progress in this genre, survival typically becomes less of a concern - environmental modifications compound to give players an increasing advantage over agents (Duncan 2011).

Though games in this genre feature villages, temples and other structures not constructed by the player, these are generated procedurally using rules or templates as a pre-processing step - the player will never encounter agents in the process of building, nor will they experience competition for resources or witness structures they destroy undergoing repair.

Designing emergence is a notoriously difficult problem (Dormans 2012), in that there is typically no guarantee that behaviors exhibited during testing will be repeated. Commercial games attempting to sell a specific experience require highly reliable AI, favoring agents which use search algorithms and behavior trees to interact with the player in well-defined ways. Independent developers have thus far favored complexity to emergence, as obtaining desired behaviors through explicit specification allows for more rapid development. The proposed framework has been envisioned as a highly modular approach which lends itself to a simple initial implementation with ample room for optimization and extension. This approach attempts to support the emergence of interesting behaviors while achieving reliability through determinism.

The Proposed Framework

The design of this framework is motivated by the empirical observation that the majority of actions performed by novice to intermediate players in games such as Minecraft or Dwarf Fortress are based on the observed or presumed proximity of other players, agents, construction sites or resources. Influence maps (Rabin 2001) were initially considered as the design primitive due a history of being used for performing spatial analysis in real-time strategy games, though the information provided by influence maps was judged to be insufficient for driving very specific behaviors. Distance maps (Rosenfeld and Pfaltz 1968) were eventually chosen as a replacement, as they are suited to grid-based worlds and allow nearest neighbor query costs to be amortized over multiple frames - furthermore, if distance maps exist for every type of object, a transformation of weighted distances will provide an influence value.

While distance maps alone provide agents with sufficient information to move between objects, allowing for a surprising variety of behaviors, there may be situations where designers wish to specify points of interest in the world, in terms of absolute locations or criteria. Hierarchical distance maps provide alternate representations of the world by allowing designers to specify functions which may utilize the data in other distance maps in order to determine which locations in the world constitute a goal. Hierarchical distance

*emory.myers@digipen.edu
†pushpak.karnick@digipen.edu

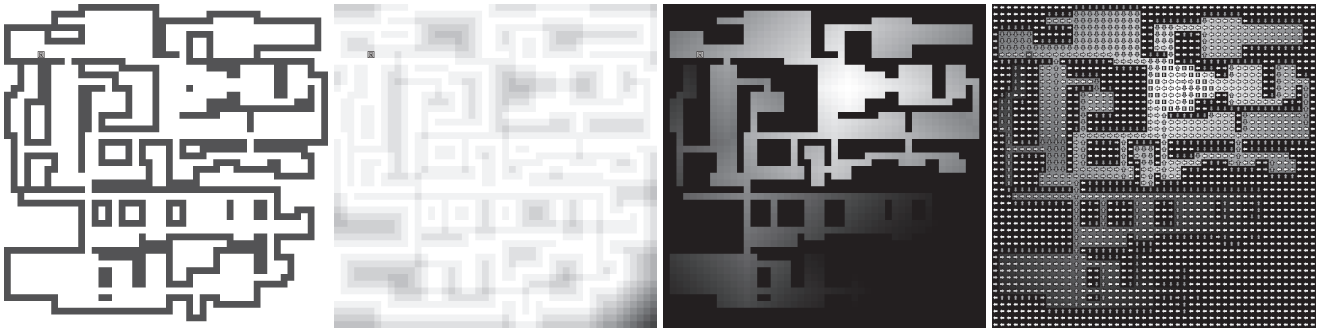

Figure 1: Our method illustrated with an example. *(From left to right)*: (a) An example map from a typical game scenario. This map contains farms and walls enclosing the farms. (b) The distance map to the walls in the scene. (c) The distance map to the farms. (d) The visualization of potential behavior paths for the agent (marked with an @) based on the combination of both distances.

mapping allows individual behaviors to remain simple and portable, while still allowing designers to achieve very specific results which otherwise may not be possible within the framework.

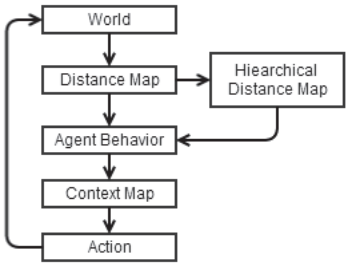

Figure 2: The structure of the proposed framework

Distance Mapping

One of the benefits of using distance maps as part of this framework is the variety of algorithms capable of creating and updating them and the trade-offs these present for developers in terms of implementation effort, accuracy and performance. Unlike the traditional distance mapping used for image analysis, which is concerned with the distance from within a shape to the boundary (Rosenfeld and Pfaltz 1968), our use case is concerned with path distances between the current position and the closest source - sources, in this case, being objects of the type the distance map corresponds to. As we're interested in obtaining path distances, we also need to ensure obstacles (objects impeding movement, such as walls) are taken into account.

Among the easiest methods for creating and updating a distance map is a modified brushfire algorithm (Blum 1967), which on an obstacle-free map would require two passes. While the brushfire transform is easy to implement (see Algorithm 1) it becomes highly inefficient in the presence of obstacles, with the number of passes depending on the map's configuration and the time complexity of a single pass being $O(2n^2)$. Thus, while sufficient for initial results, this approach is not recommended for serious usage.

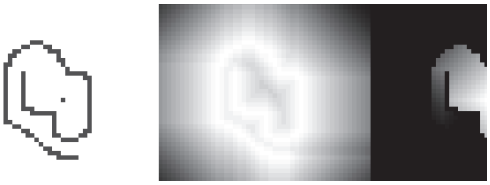

Figure 3: The game map, the distance map for walls and the distance map for farms

Fast marching (Tsitsiklis 1995), a class of algorithm based on wave front propagation, allows for the creation of a distance map in a single iteration in $O(\mathrm{n}log\mathrm{n})$ without experiencing performance degradation on maps containing obstacles. Though a bit more complex, the inner-workings of a fast marching approach are very similar to using Dijkstra's algorithm for shortest path calculations, easing the burden of implementation for developers familiar with A*. Through the introduction of a data structure called the untidy priority queue (Yatziv, Bartesaghi, and Sapiro 2005), which uses buckets similarly to a hash table, priority queue insertion times can be reduced to $O(1)$, improving the performance of fast marching to $O(n)$.

Constructing a distance map is integral, but given that this approach is tailored to dynamic environments the ability to update distance maps in realtime, though not necessarily on a per-frame basis, is also required. Similar to the wavefronts used when creating a distance map via fast marching, distance maps may be updated by the propagation of "upper" and "lower" wavefronts created via map modifications (Lau, Sprunk, and Burgard 2010) to significantly reduce the amount of computation necessary by minimizing the number of distance recalculations.

Distance Metrics and Aesthetics

The selection of a distance metric used in a particular distance map will have a strong impact on all calculations us-

Algorithm 1 A modified brushfire algorithm

for all $x, y \in sources$ **do**
  $d(x, y) \leftarrow 0$
end for
for all $x, y \in obstacles$ **do**
  $d(x, y) \leftarrow max$
end for
for $y = 0 \rightarrow n$ **do**
  for $x = 0 \rightarrow m$ **do**
    if $map(x, y) \notin sources \cup obstacles$ **then**
      $i \leftarrow d(x - 1, y) \wedge d(x + 1, y)$
      $j \leftarrow d(x, y - 1) \wedge d(x, y + 1)$
      $d(x, y) \leftarrow ((i \wedge j) + 1) \wedge max$
    end if
  end for
end for
for $y = n \rightarrow 0$ **do**
  for $x = m \rightarrow 0$ **do**
    if $map(x, y) \notin sources \cup obstacles$ **then**
      $i \leftarrow d(x - 1, y) \wedge d(x + 1, y)$
      $j \leftarrow d(x, y - 1) \wedge d(x, y + 1)$
      $d(x, y) \leftarrow ((i \wedge j) + 1) \wedge max$
    end if
  end for
end for

ing that information. The work thus far has verified that construction resulting from a distance map will resemble the gradient created by the distance metric (see Figure 4.) Using 4-connected topologies will result in diamond-shaped features, 8-connected topologies will favor square features and utilizing actual euclidean distance will favor circular constructions.

Visualization Design

Distance maps and hierarchical distance maps lend themselves to being viewed as gradients by a direct mapping of distances to brightness. Individual agent behaviors populate a context map which contains, for every action an agent may perform, floating point values for interest and danger between 0.0 and 1.0. Populating a context map using multiple behaviors provides all the information necessary to visualize how an agent using that combination would behave. By overlaying imagery corresponding to the action an agent would select at every position in the world (see Figure 4,) it is possible to rapidly understand how a change to a single behavior will impact the aggregate behavior of an agent in many different circumstances. Such a visualization would benefit the designer by providing a global view of the choices available to the agent and an understanding of which behaviors led an agent to performing a particular action.

Keeping agent state minimal puts most of the complexity into the behaviors, while keeping those behaviors deterministic (based only on the current state of the map and an agent's inventory) allows for visualizations to be reliable.

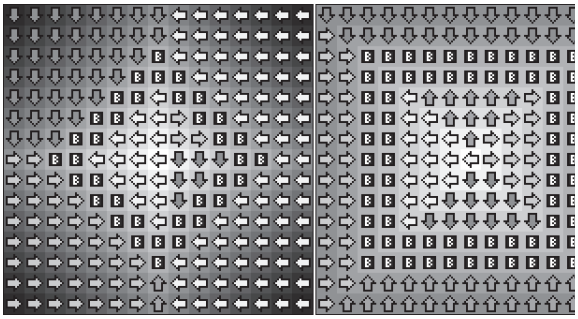

Figure 4: Visualizations of a wall building behavior on a 4-connected topology (left) and an 8-connected topology (right.)

Related Work

Craig Reynolds' work with steering behaviors (Reynolds 1999) serves as one of the inspirations for this approach, as it showed that the result of combining relatively simple behaviors in an agent-based simulation could give rise to more advanced behavior being exhibited by the group as a whole. As Reynolds' behaviors were only concerned with returning a vector used to steer an agent's locomotion, a simple linear combination often worked well, with behavior like flocking emerging from such a combination of alignment, cohesion and separation behaviors.

In the years since Reynolds' paper, developers have struggled to successfully use this approach in games - steering behaviors are prone to lead agents into local minima which typically eliminates them from consideration in complex game environments. Context behaviors (Fray 2013) seek to expand the potential of behavior-based systems by introducing the notion of behavior context via context maps. Context maps, as described by Fray, consist of two 1D arrays representing interest and danger. Positions in these arrays are referred to as slots and correspond with an action to be performed by the agent. Behaviors only write to context slots if that value would exceed the existing value. After the context map is populated, the action to be performed is determined by choosing the slot with the maximum interest from among the slots with the minimum danger. Although Fray introduced context in order to simplify steering behaviors, by mitigating the need to avoid local minima in individual behaviors, context maps are also suitable for the development of behaviors which need to return information other than a steering direction.

Flow field pathfinding (Treuille, Cooper, and Popovic 2006) has been successfully used in games such as Supreme Commander 2 and is based on techniques developed for the simulation of crowds in cities. Relying on two levels of information, flow field pathfinding is attractive in that computational effort is spent on updating the underlying representations rather than performing searches for individual agents. At the upper level, multiple flow fields are calculated using typical pathfinding algorithms in order to provide directions between goals. At a lower level, flow fields are created using artificial potential fields to calculate vectors which provide

local obstacle avoidance. Both layers are then combined to form a final flow field on which agents perform simple lookup operations, allowing the approach to scale from simulating a handful of agents to entire crowds.

Existing work on procedural city modeling (Lechner, Watson, and Wilensky 2003) has demonstrated that a realtime approach relying on conceptually simple behaviors lends itself to city planning at a high level. An agent-based simulation was shown to be capable of building road networks and zoning adjacent properties in ways which resemble actual cities, with small changes in behavior weights leading to designs approximating both planned and organic city growth. This work showed that behaviors driven by the state of the world, which also modify the world, may lead to agents which produce aesthetically pleasing results while appearing to cooperate without inter-agent communication having been explicitly provided.

Proposed Behaviors

This section contains an overview of construction behaviors and how they would benefit from the proposed framework. For the purposes of this overview the terms object and resource are used interchangeably to describe both traditional game objects, such as swords, as well as types of terrain, such as limestone. In block world games the line between objects and terrain is blurred, with impassible objects such as furnaces often stored as terrain. For the purposes of distance map construction, items and terrain are treated identically, and so can be used interchangeably when authoring construction behaviors.

Resource Gathering

Resource gathering, in a broad sense, describes all behaviors where the goal is having an agent move to a position adjacent to an object and remove it from the world, with the object usually being moved into an agent's inventory rather than destroyed. Resource gathering is an important activity in the block-world genre as many actions depend on agents having access to objects at arbitrary positions. The existence of a separate distance map for every type of object allows agents to move towards a desired resource by descending the gradient in the appropriate distance map. The magnitude of the context values supplied by a resource gathering behavior are proportional to both the amount of that resource in the agent's inventory as well as the distance between an agent and that type of resource. Combining multiple resource gathering behaviors allows for the emergence of distraction, where agents moving toward a scare resource may begin gathering a more abundant resource along the way. Distraction may be observed in Minecraft by human players, who routinely collect objects of minimal utility, such as flowers, while moving between sites containing more valuable resources, such as wood.

Wall Building

Wall building describes behaviors where the goal is to separate a single space into two or more areas. Walls are useful for isolating sources of danger, such as a troll sleeping in a cave, or for protecting agents during periods of vulnerability, such as while they are asleep. Wall building in a block-world game is similar to wall building in an RTS, where several algorithms have been researched including a relatively simple distance-based approach described by Mario Grimani (Grimani 2004). For the purposes of protection or isolation, wall building behaviors can take advantage of object-based distance maps to quickly identify all positions within a specified distance range. The construction of walls using distance maps alone leads to situations where agents without another behavior to distract them may become trapped in local minima, having built a wall and found themselves on the wrong side of it. Hierarchical distance maps, when utilized for wall building, are better suited for agents who would routinely become trapped or when walls should be constructed based on criteria which cannot be sufficiently expressed in terms of object distances.

Door Placement

Door placement is a behavior which seeks to connect adjacent spaces without merging them, for the purpose of moving between them. Doors are very useful for allowing sentient creatures to travel between connected areas, such as the outdoors and a hut, while preventing animals from doing the same. Doors can also be designed to support authentication-based unlocking mechanism that depends on the in-game status of the sentient characters. In order for the proposed framework to support door placement, a connected component labelling algorithm is used which identifies separate areas - this algorithm is modified to also keep track of which areas are linked together by doors. Door placement relies on a hierarchical distance map which identifies narrow walls between unlinked areas as candidate buildsites. Combining wall building and door placement will allow for the emergence of constructions which are recognizable as buildings.

Road Construction

Road construction is a behavior which seeks to join points of interest, such as external doors, with roads or paths. Roads are important in that they serve to decrease the time required to travel by increasing movement speeds. Roads also serve an important function for players in games where visibility is limited, in that following a road generally guarantees arriving at an interesting destination. The proposed behavior for road construction mirrors the approach used by Lechner et. al. when constructing roads for a city simulation (Lechner, Watson, and Wilensky 2003). This method favors starting the simulation with a small road network and subsequently modifying and expanding the network as the simulation demands.

Tunneling

Unlike door placement, which seeks to link areas, tunneling removes objects and terrain in order to merge two areas into one. Tunnels are important for situations where roads are impractical because the distance between points of interest is unacceptably high due to the presence of obstacles. Tunneling may be a special case which relies on a hierarchical

distance map populated by a traditional search algorithm, such as A*. With the locations to tunnel out having been designated, it is possible that agents may simultaneously begin tunneling from both sides of the obstacle. As tunnels may expose resources, such as coal veins, agents with tunneling and resource collection behaviors may appear to be mining, without a mining behavior having been explicitly assigned to them.

Mining

Mining is a behavior which seeks to dig tunnels or create hollow areas in natural obstacles, not for the purpose of travel, but also to discover and harvest useful resources. Mining is important because it is generally the only way to advance through the technology tree in block world games, as the resources necessary for crafting more powerful objects are initially unreachable. Although mining may emerge from a combination of tunneling and resource collection, an explicit mining behavior will be necessary as tunneling is not guaranteed to expose resources. If reliability were not a concern, mining would be a relatively simple behavior which consisted of an agent performing a random walk, with some amount of bias for their previous direction controlling the smoothness of the path describing the resulting tunnels. Using stochastic behavior as an inspiration, an agent would examine its surrounding area to determine the next most likely direction for movement (in order to determine a bias) and then select a new direction in which to mine based on the interaction of that bias with a direction derived from their current position.

Object Placement

Object placement is a broad category of behaviors which utilize distance maps to place objects in the world, often in relation to other objects and in response to absence of any object of that type within a specified distance. Examples of object placement behaviors would include farming, where the objects to be placed are seeds or saplings, or furnishing, where a bed may be placed if there are no other beds available nearby which are unoccupied. Object placement behaviors are among the most important, as one agent placing an object is extremely likely to affect which action is selected by nearby agents. Furnishing, when combined with wall building and door placement, should be responsible for creating houses.

Crafting

Crafting behaviors allow agents to transform objects in their inventory into other objects, a process which is usually irreversible and may require the utilization of an object in the world. Crafting behaviors are important because, in the block world genre, some resources require specialized tools in order to be collected - crafting behaviors allow agents to transform those resources they are able to collect into objects which allow the collection of further resources. Crafting behaviors also allow agents to transform a raw resource into another type of resource, which may be necessary for further crafting to take place. An example of a crafting behavior would be iron smelting, which requires the agent has iron ore in their possession and is within some specified vicinity of a furnace. Crafting behaviors are driven by resource collection, which is handled elsewhere, as well as the ability to move adjacent to objects in the world, which may also be driven by performing gradient descent in the appropriate distance map. As crafting behaviors modify the agent's inventory, other behaviors which take inventory contents into consideration will be greatly affected.

Discussion

To summarize, this paper proposes a method for the development of agents in the block world genre. The approach described is based on behaviors which populate a context map, from which the agent selects an action to perform using well-defined criteria. Individual behaviors are based on obtaining distance values from distance maps, which exist for each type of object, as well as hierarchical distance maps, where positions with a value of 0 are referred to as sources and specified by a designer. This paper illustrates how context maps, when populated by deterministic behaviors, allow for robust visualization of emergent agent behavior. Lastly, this paper discusses potential applications of this approach by detailing several types of behaviors, along with brief discussions of their importance and remarks on how they may be expected to interact.

As this research is currently underway, there are multiple aspects of this approach which we believe may warrant further investigation:

- While up-to-date influence maps are more valuable to agents, frequently in games there are inherent delays - actions take time. While updating distance maps on a per-frame basis would be desirable, using large numbers of distance maps, or distance maps which cover large areas, may require too much time, even when using the optimized algorithm described by Lau (Lau, Sprunk, and Burgard 2010). An attractive option for game development would be moving the distance map calculations to the GPU - we have already found some research in this direction, using polygon rasterization to quickly compute generalized Voronoi diagrams (Sud et al. 2007). With the rise in availability of GPGPU-capable graphics cards in PCs and home consoles, we feel that a compute-based approach may be worthwhile.
- Although GPGPU will be suitable for utilizing the described approach in single-player games, cloud computing presents an alternative for multiplayer games, where the information required for creating distance maps may be unavailable to clients in order to prevent cheating.
- One further advantage of an approach based on distance maps is that a clearance can quickly be calculated at any position (Harabor 2009), which makes the framework reasonably well suited for supporting variable-sized agents. Supporting variable-sized agents would likely require other changes to the framework, however, as behaviors which rely on gradient descent will likely lead large agents into choke points.
- In addition to the aformentioned distance metrics, it may be possible to influence the shape of constructions and

features by utilizing designer-specified unsigned distance functions to control how distance is propagated. Using unsigned distance functions in conjunction with distance maps relying on path distance is an area where further research is needed to determine feasibility.

- While the intention of this framework is to allow for emergent behavior without individual agents performing ad hoc search, a number of interesting possibilities arise when distance maps and hierarchical distance maps are combined with traditional search. Incorporating the context map of an agent into the heuristic function in a pathfinding or planning algorithm results in a hybrid approach which may be especially suitable for developers wishing to used the proposed framework as part of a level of detail system.

References

Blum, H. 1967. A Transformation for Extracting New Descriptors of Shape. In Wathen-Dunn, W., ed., *Models for the Perception of Speech and Visual Form*. Cambridge: MIT Press. 362–380.

Dormans, J. 2012. *Engineering Emergence: Applied Theory for Game Design*. Universiteit van Amsterdam [Host].

Duncan, S. C. 2011. Minecraft, Beyond Construction and Survival. *Well Played* 1(1):1–22.

Fray, A. 2013. Context behaviors know how to share.

Grimani, M. 2004. Advanced wall building for rts games. In Kirmse, A., ed., *Game Programming Gems 4*. Charles River Media. 365–372.

Harabor, D. 2009. Clearance-based pathfinding and hierarchical annotated a* search.

Lau, B.; Sprunk, C.; and Burgard, W. 2010. Improved updating of euclidean distance maps and voronoi diagrams. In *IROS*, 281–286. IEEE.

Lechner, T.; Watson, B.; and Wilensky, U. 2003. Procedural city modeling. In *In 1st Midwestern Graphics Conference*.

Lynley, M. 2013. 'Minecraft' Hits Mother Lode For a Small Swedish Company. *The Wall Street Journal* B1.

Rabin, S. 2001. Strategies for Optimizing AI. *Game Programming Gems 2* 251–257.

Reynolds, C. 1999. Steering behaviors for autonomous characters.

Rosenfeld, A., and Pfaltz, J. L. 1968. Distance functions on digital pictures. *Pattern Recognition* 1(1):33–61.

Sud, A.; Govindaraju, N. K.; Gayle, R.; Andersen, E.; and Manocha, D. 2007. Surface distance maps. In *Graphics Interface*, 35–42.

Treuille, A.; Cooper, S.; and Popovic, Z. 2006. Continuum crowds. *ACM Trans. Graph* 25:1160–1168.

Tsitsiklis, J. N. 1995. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control* 40:1528–1538.

Yatziv, L.; Bartesaghi, A.; and Sapiro, G. 2005. O(n) implementation of the fast marching algorithm. *Journal of Computational Physics* 212:393–399.