# The A-Maze-D Advanced Maze Development System for Fast Game Design and Implementation

Shruti Daggumati, Peter Z. Revesz, and Corey Svehla
Department of Computer Science and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska 68588
sdagguma@cse.unl.edu, revesz@cse.unl.edu, csvehla@cse.unl.edu

***Abstract*—**The medium of computer and video games are being studied profusely in a wide array of research. Constraint databases and video games have been around for a long time, but they have never been combined together before. In this paper, we describe the A-Maze-D system, which shows that constraint databases can be applied conveniently and efficiently to the design of maze games. A-Maze-D provides a versatile set of features by the combination of MATLAB scripts and the MLPQ constraint database system. A-Maze-D enables an efficient implementation of many complex features existing in games such as collision and hindering vision. A-Maze-D is the first system that uses constraint databases to build maze games and opens new ideas in video game development.

***Index terms*—** animation, constraint database, maze, MLPQ, moving objects, video game.

## I. Introduction

Video games have become ubiquitous on our computers, consoles and phones. The rapidly growing video game industry has a revenue of approximately twelve billion U.S. dollars per year in the United States alone according to *Statista*, the statistics Internet portal (http://www.statista.com/statistics/201093/revenue-of-the-us-video-game-industry/). The expanding video game industry demands the efficient development of new video game products.

A large set of video games require the representation of a map, usually some kind of maze, and other spatial objects. In addition, video games also routinely require the representation of moving objects. Hence video games have a strong connection with geographic, spatial and moving object (also called spatio-temporal) databases. Since these types of databases can be viewed as special cases of constraint databases (Kanellakis et al. [10], Revesz [18]), we propose *A-Maze-D*, an *Advanced Maze Development* system with a novel design based on the MLPQ system [21].

Our proposed A-Maze-D system provides a large set of useful features that enable game development where the main objective is to find the way out of a maze with limited viewing distance from an overhead view. We describe in detail the features that are the most important in developing maze games. We envision that with a growing set of features, the development can be extended from maze games to a plethora of different types of video games.

This paper is organized as follows. Section II describes some related work on mazes and constraint databases. The next three sections present various aspects of the A-Maze-D (Advanced Maze Development) system. In particular, Section III shows how A-Maze-D can specify stationary objects such as mazes with either straight or curved walls. Section IV describes the specification of moving objects such as persons, exploding objects and shields. Section V describes the interaction of various objects such as shield type objects blocking bullets or explosions and elastic objects colliding with each other and then bouncing back from each other. Finally, Section VI provides some conclusions and possibilities for future work.

## II. Related Work

A maze is a complex passage with multiple branches where the user needs to find the best route (Matthews [14]). The subject of mazes occurs in many places in biology, psychology and video games. Section II-A gives a brief history of mazes and video games development focused on games that involve mazes. Section II-B reviews constraint databases that are used by the A-Maze-D system.

### A. Mazes and Maze Games

In most mazes, the walls are fixed and do not change as the user progresses through the maze. Maze solving computer algorithms that try to find the best path to an exit or the fastest way to attain a prize have been implemented many times. However, the typical solutions do not take into account a user's limited vision and inability to mark the walls as hindrances for solving the mazes. In Section 3, we specifically allow the option of representing the limited vision of a user.

In games like Super Mario World, there are levels where the user needs to solve a maze and has limitations as to what they can see. This type of maze occurs in many different games including some Legend of Zelda games. In addition, in video games such as Super Mario World if the user has failed a level numerous times, then the system offers the ability for the system to show the proper way to finish the level where obstacles of all kinds are taken into account and the shortest path is used.

In well-known games such as Pac-Man, the objective is to collect all the pellets and to live as long as the ghosts do not eat the user. In other games, the goal is to rescue a princess or to find the treasure and the end of the journey. In the implemented

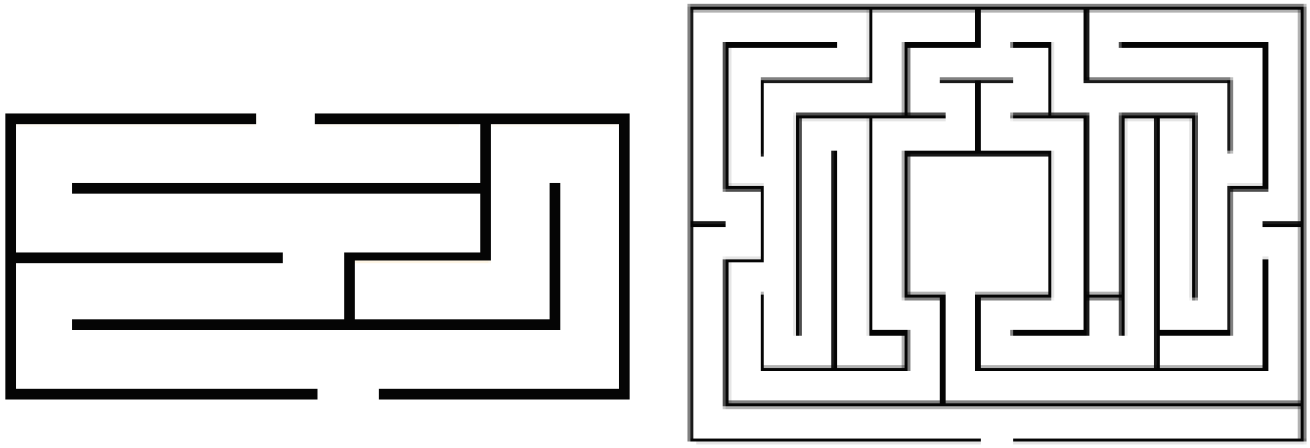


Fig. 1: Examples of mazes with straight walls. The one on the left has an entrance at the bottom and an exit on the top. The one on the right has an entrance on the bottom and the goal is to reach the large central square.

examples in this paper, we favor the idea of there being some form of treasure at the end of the levels.

Mazes form also an important element in biology in the study of learning. Rodents were first used in mazes by Willard Small [25]. Using rodent burrows as a design a maze was created to test the cognitive abilities of rodents. Soon after these early experiments, different animals were used by James Watson for testing purposes ranging from monkeys to birds [26]. Watson also sent rodents through a maze with some sensory deprivations. A few years later, Fleming Perrin tested humans where he blindfolded each person and let them solve a dodecagonal maze [17]. Many of these types of experiments can be modeled using maze choice games.

Some current tools used by game developers to make games is described in [12]. Herwig and Paar [8] describes an approach of using game engines for landscape visualization and planning.

Eberly [7] and Millington [15] review the physical aspects of games, such as the physics behind collisions and explosions. Kriegel et al. [11] describe a multi-layer games management system where players can explore large maps. Rogers [24] describes an alternative approach to level design in multi-layer games.

### *B. Constraint Databases*

Constraint databases were introduced by Kanellakis et al. [10]. Constraint databases provide an extension of relational databases where the input data may be an infinite size relations that are finitely represented by mathematical constraints [18]. This finite representation is achieved by making each constraint relation a set of constraint tuples. In constraint tuples, the attributes are referred to by variables and the possible values of the attribute variables are restricted by constraints. Constraint databases developed in many directions with a diversity of applications. Revesz [18] is an introductory database textbook that covers constraint databases and their applications.

The MLPQ system, developed at the University of Nebraska-Lincoln, is one of the systems that implements and visualizes constraint databases. Constraint databases have been used already in many spatial and moving object database applications, including monitoring the spread of epidemics [23], congestion forecasting by MaxCount, an operator that estimates the maximum number of moving objects that could in an area [1], spatio-temporal interpolation [13], spatial and temporal data mining [22] including predicting pancreatic cancer based on cancer-related genes and their temporal patterns of gene expressions in the cells of pancreatic cancer patients [20].

## III. Implementation of Stationary Objects

The A-Maze-D, short for Advanced Maze Developer, system is a versatile system that enables efficient development of complex maze games. The A-Maze-D system consists of a MATLAB library that allows the easy translation of input data into MLPQ system input files. Once all stationary and moving objects are translated into MLPQ input files, the input files can be opened and animated in the MLPQ system. The animation lasts until some choice point is reached. The choice point requires that the user enter some input parameters, such as whether to turn left or right at the current location in the maze, to fire some bullets, to start a conversation or some other action. Once the input parameters are entered the corresponding animation can continue until the next choice point.

The main stationary object in a maze game is the maze itself. Mazes can be divided into two different types. The first type of mazes have only straight walls, while the second more complex type of mazes have also curved walls. The implementation of mazes, especially with curved walls, can be a tedious software engineering task. However, we show below that using the A-Maze-D system mazes can be developed efficiently whether the mazes have straight or curved walls.

### A. Mazes with Straight Walls

Figure 1 shows two mazes with walls that are composed of straight lines. Suppose that we would like to implement the maze shown on the left side of Figure 1. At first, we record the corner points for each wall as shown with highlights in Figure 2.

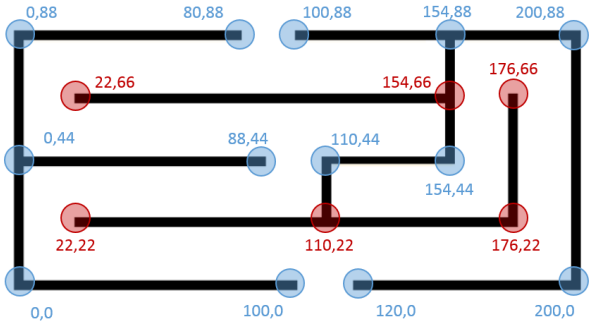



Fig. 2: Examples points on a maze. The colors of the points are used to show which points are connected with each other to form a single wall.

Each wall needs to have a different id. In the case of the maze in Figure 2 we need six walls with the $(x, y)$ corner points shown in Table I. Each wall has a separate id number, which appears in the first column of Table I. The id numbers in Table I describe a way of drawing the maze in Figure 2 by using only six walls, but other ways of drawing the same maze are possible.

TABLE I: The $(x, y)$ coordinates of the corner points on the maze. This data is used as input to the A-Maze-D system to develop the maze. Each wall has a separate id number.

| id | X | Y |
|---|---|---|
| 1 | 100 | 0 |
| 1 | 0 | 0 |
| 1 | 0 | 88 |
| 1 | 80 | 88 |
| 2 | 120 | 0 |
| 2 | 200 | 0 |
| 2 | 200 | 88 |
| 2 | 100 | 88 |
| 3 | 0 | 44 |
| 3 | 88 | 44 |
| 4 | 22 | 22 |
| 4 | 176 | 22 |
| 4 | 176 | 66 |
| 5 | 110 | 22 |
| 5 | 110 | 44 |
| 5 | 154 | 44 |
| 5 | 154 | 88 |
| 6 | 22 | 66 |
| 6 | 154 | 66 |

A-Maze-D has a library of MATLAB scripts that contains a function called *buildWalls* that takes as input the set of points and turns them into an MLPQ input file that represents the walls. The basic idea behind the *buildWalls* function is illustrated in Figure 3, which shows on the left a simple table with two points and on the right their implied meaning. The *buildWalls* function also needs a parameter that specifies the width for the wall. In this case for simplicity we chose a width of one. As can be seen, the $(a, c)$ and $(b, d)$ points define a parallelogram with width one and whose lower boundary line is the line segment from $(a, c)$ to $(b, d)$.

The *buildWalls* function transforms the points described in Table I into the following MLPQ input file:

```
begin %MLPQ%
R(id,x,y) :- id=1, x>=0, x<=100, y=0.
R(id,x,y) :- id=1, x=0, y>=0, y<=88.
R(id,x,y) :- id=1, x>=0, x<=80, y=88.
R(id,x,y) :- id=2, x>=120, x<=200, y=0.
R(id,x,y) :- id=2, x=200, y>=0, y<=88.
R(id,x,y) :- id=2, x>=100, x<=200, y=88.
R(id,x,y) :- id=3, x>=0, x<=88, y=44.
R(id,x,y) :- id=4, x>=22, x<=176, y=22.
R(id,x,y) :- id=4, x=176, y>=22, y<=66.
R(id,x,y) :- id=5, x=110, y>=22, y<=44.
R(id,x,y) :- id=5, x>=110, x<=154, y=44.
R(id,x,y) :- id=5, x=154, y>=44, y<=88.
R(id,x,y) :- id=6, x>=22, x<=154, y=66.
end %MLPQ%
```

### B. Mazes with Curved Walls

Figure 4 illustrates a maze with curved walls. The A-Maze-D system uses multiple MATLAB scripts in order to attain the smoothest-looking curved walls. For example, Figure 4 shows a maze where all walls are curved except for six straight walls that are all horizontal and are used to block further passage in maze at dead ends.

Let us illustrate the A-Maze-D functions using the bottom curved wall, which is a segment of the parabola $y = (x/4)^2$. An approximation of the parabola can be specified by eleven distinct points as shown in Figure 4. Our MATLAB script takes the $x$ coordinates and the parabolic function $y = (x/4)^2$ to generate Table II. The script can take any other polynomial function. In case the boundary of the wall cannot be described by the user as a polynomial function, the A-Maze-D system also provides an alternative MATLAB script that makes a cubic spline interpolation for the given sample points of the wall.

The more points we choose for the approximation, the smoother-looking parabolic curved wall we obtain. However, the smoother representation generates a larger MLPQ input file, which means generally a slower visualization and animations of the maze game. Given the above input data, the A-Mazed-D system already generates a large MLPQ output file, which can be seen in Figure 5. We have also included our MATLAB implementation that was created specifically to build curved walls and then to output the proper format of each curved wall into a MLPQ format. The code can be seen in Figure 6. This MATLAB script uses a the built-in function

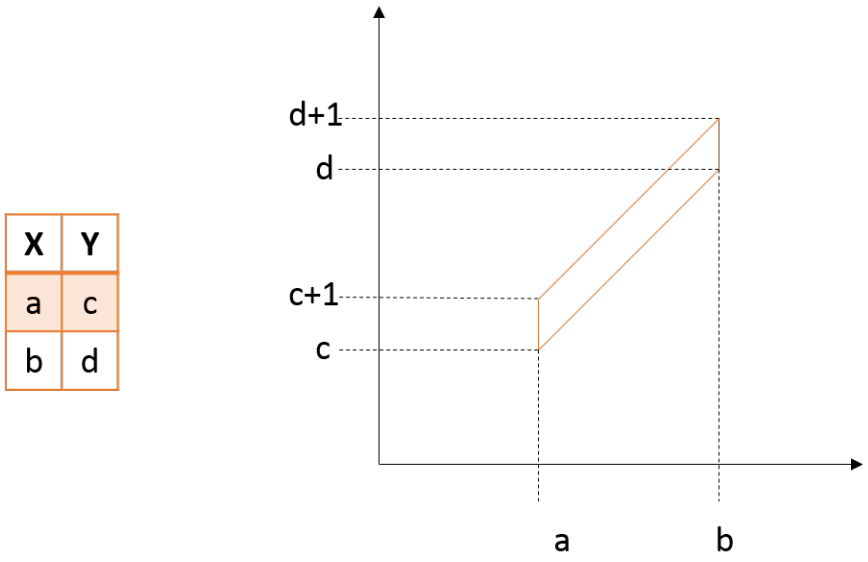



Fig. 3: The points from the table are represented visually on the right.

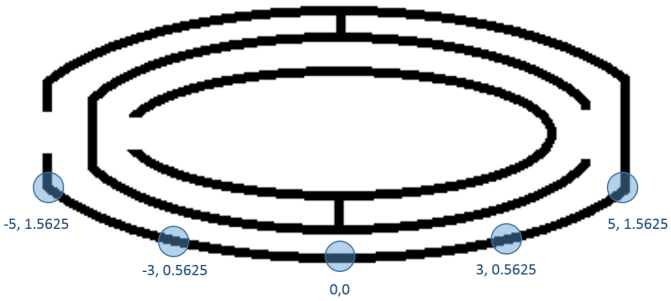



Fig. 4: Example of a maze with curved walls and selected sample points on the bottom curved wall.

TABLE II: Example corner points on a maze.

| id | X | Y |
|---|---|---|
| 1 | -5 | 1.5625 |
| 1 | -4 | 1 |
| 1 | -3 | 0.5625 |
| 1 | -2 | 0.25 |
| 1 | -1 | 0.0625 |
| 1 | 0 | 0 |
| 1 | 1 | 0.0625 |
| 1 | 2 | 0.25 |
| 1 | 3 | 0.5625 |
| 1 | 4 | 1 |
| 1 | 5 | 1.5625 |

`spline`, which computes for a set of points the piecewise cubic polynomial interpolation for a set of points [2], [19]. Although MATLAB does not explicitly return the piecewise cubic polynomial, the MATLAB script takes a set of sample values from the interpolation. When the number of sample values is large enough, then the piecewise linear interpolation among those points will generate a smooth-looking curve.

### C. Color Change

We call *color animation* when stationary or moving objects change color over time. Color animation can provide important visual cues to the users. For example, in our maze we could make the walls change color that would show some type of time limit where the walls will start to change to a specific color when the user is running out of time. MLPQ handles color animation by allowing the user to choose two different colors for each relation.

When an MLPQ input file is loaded into the MLPQ system, then next to each spatial or moving object relation there are two colored boxes as shown in Figure 7. The first box represents the starting color, and the second box the ending color for the displayed relation during the animation. The MLPQ system provides a random pair of colors for each relation after a new file is uploaded. To change the colors to the desired values, we can double click on one of the boxes. Then a color scheme window will pop up as shown in Figure 8. This window allows the user to create the specific color of their choice for the selected box. For example, Figure 9 shows two snapshots of the color animation of a maze, which changes from green to orange.

## IV. Implementation of Moving Objects

### A. Player Movement

Movement of the player is guided by choices that the player can select in MLPQ. When the player chooses a movement option in the maze, the A-Maze-D system animates the player moving through the maze until the player reaches the next choice point where another decision is required. At any moment in time, each player is assumed to occupy a $4$ by $4$ square area. The movement of the square is represented by the attributes $x$, $y$ and $t$, which denoted time. For a moving

```
begin %MLPQ%
R(id,x,y) :- id=1, x>=-4.000000, x<=-3.920000, -0.640043x - y=1.560172.
R(id,x,y) :- id=1, x>=-3.920000, x<=-3.840000, -0.588340x - y=1.357495.
R(id,x,y) :- id=1, x>=-3.840000, x<=-3.760000, -0.541344x - y=1.177033.
R(id,x,y) :- id=1, x>=-3.760000, x<=-3.680000, -0.499057x - y=1.018034.
R(id,x,y) :- id=1, x>=-3.680000, x<=-3.600000, -0.461478x - y=0.879744.
R(id,x,y) :- id=1, x>=-3.600000, x<=-3.520000, -0.428608x - y=0.761409.
R(id,x,y) :- id=1, x>=-3.520000, x<=-3.440000, -0.400445x - y=0.662276.
R(id,x,y) :- id=1, x>=-3.440000, x<=-3.360000, -0.376990x - y=0.581592.
R(id,x,y) :- id=1, x>=-3.360000, x<=-3.280000, -0.358244x - y=0.518604.
R(id,x,y) :- id=1, x>=-3.280000, x<=-3.200000, -0.344206x - y=0.472559.
R(id,x,y) :- id=1, x>=-3.200000, x<=-3.120000, -0.334876x - y=0.442702.
R(id,x,y) :- id=1, x>=-3.120000, x<=-3.040000, -0.330254x - y=0.428282.
R(id,x,y) :- id=1, x>=-3.040000, x<=-2.960000, -0.330215x - y=0.428165.
R(id,x,y) :- id=1, x>=-2.960000, x<=-2.880000, -0.331900x - y=0.433151.
R(id,x,y) :- id=1, x>=-2.880000, x<=-2.800000, -0.332445x - y=0.434722.
R(id,x,y) :- id=1, x>=-2.800000, x<=-2.720000, -0.331727x - y=0.432712.
R(id,x,y) :- id=1, x>=-2.720000, x<=-2.640000, -0.329746x - y=0.427324.
R(id,x,y) :- id=1, x>=-2.640000, x<=-2.560000, -0.326502x - y=0.418760.
R(id,x,y) :- id=1, x>=-2.560000, x<=-2.480000, -0.321995x - y=0.407221.
R(id,x,y) :- id=1, x>=-2.480000, x<=-2.400000, -0.316225x - y=0.392911.
...
end %MLPQ%
```

Fig. 5: Defining a curved maze as an MLPQ input file.

```
function buildCurvedWallsLatex(id,x,y)
   % x and y are the list of x and y coordinates, respectively,
   % of successive sample points of the curved wall.
   %Example:  x = [-5 -4 -3 -2 -1 0 1 2 3 4 5] and y = (x/4).^2
    cs = spline(x,[0 y 0]);
    xx = linspace(-4,4,101);
    pp = ppval(cs,xx);
    printWalls(id,xx,pp)
end

function printWalls(id,x,y)
    fid1=fopen('result.txt','wt');
    fprintf(fid1, 'begin %%MLPQ%% \n');

    for i=2:length(x)
        xs=[x(i-1),x(i)];
        ys=[y(i-1),y(i)];
        mb = polyfit(xs,ys,1);
        fprintf(fid1, 'R(id,x,y) :- id=%d, x>=%f, x<=%f, %fx - y=%f.\n',
            id,x(i-1),x(i),mb(1),-mb(2));
    end

    fprintf(fid1, 'end %%MLPQ%% \n');
    fclose(fid1);
end
```

Fig. 6: Code for creating and displaying the curved walls.

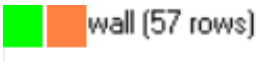



Fig. 7: An example of specifying color change where during animation the beginning color is on the left and the ending color is on the right.

object, $x$ and $y$ are functions of time represented by linear constraints in the MLPQ system input files. The A-Maze-D system uses another MATLAB script to generate the MLPQ file.

The MATLAB script will automatically compile this and return the movements for your player to take. Once the constraints are created from the script you then have to put them in your MLPQ maze file with the proper relations. The code below is one example that shows how the constraints are stored inside the MLPQ file.

For example, Figure 10 shows the movement of the player from location $(110, 110$ to location $(99, 55)$. Besides these beginning and ending points, we also record some of the other points that are on the way with the restriction that all turning points need to be included in the list. In this case, the selected points can be represented as shown in Table III. The A-Maze-D system generates the following MLPQ file for the input shown in Table III.

```
begin %MLPQ%
player(id,x,y,t) :- id=1,
                    x + t >= 119,
                    x + t <= 121,
                    y > 10, y <= 11,
                    t >= 10, t <= 109.

player(id,x,y,t) :- id=1,
                    x >= 10, x <= 12,
                    y - t > -99,
                    y - t <= -98,
                    t >= 109, t <= 131.

player(id,x,y,t) :- id=1,
                    x - t >= -121,
                    x - t <= -119,
                    y > 32, y <= 33,
                    t >= 131, t <= 219.

player(id,x,y,t) :- id=1,
                    x >= 98, x <= 100,
                    y - t > -187,
                    y - t <= -186,
                    t >= 219, t <= 241.
end %MLPQ%
```

Usign the above input file, the A-Maze-D system generates an animation using MLPQ as a basis.

TABLE III: The center points for a moving object in the maze.

| id | X | Y |
|---|---|---|
| 1 | 110 | 11 |
| 1 | 11 | 11 |
| 1 | 11 | 33 |
| 1 | 99 | 33 |
| 1 | 99 | 55 |

### B. Searchlight

The A-Maze-D system also allows a limited field of view for the game player by adding a larger square relation around the moving player. The complement of that larger square will be displayed in dark blue over the actual maze. For example, Figure 11 shows the limited field of view provided by a search light. This figure also shows how the searchlight follows the player as he or she moves in the maze. Then at the end of the maze the player has reached the treasure, and therefore has successfully completed the first level of the maze.

### C. Explosions

The A-Maze-D system can represent explosions and other expanding objects. For example, fireworks can be represented as an object that expands until it ceases to exist. The A-Maze-D system provides a function that gives as input the beginning and the ending shapes of the exploding object and gives as output a parametric rectangles representation of the expanding object that is an accepted MLPQ input file.

Color animation can be applied to moving objects too similarly as it is applied to stationary objects. For example, the firework could be black at the beginning and gradually lighted up and become completely red in the end.

## V. Interactions of Objects

The A-Maze-D system allows stationary and moving objects to interact with each other in many different ways. We describe some of the possible interactions.

### A. Explosions and Shields

The A-Maze-D system allows a shield type object to prevent the further expansion of an exploding object. For example, Figure 13 (a) shows a room with surrounding walls (green), a column (blue horizontal rectangle) and a grenade (small black square). The walls and the column are stationary objects, while the grenade is an exploding object. Figure 13 (b) and (c) show two snapshots of the A-Maze-D system animation of the exploding grenade. The animation shows the grenade explosion slowly expanding and turning red and being blocked by the wall and the column. Explosions and shields in the A-Maze-D system are implemented using the MLPQ block operator.

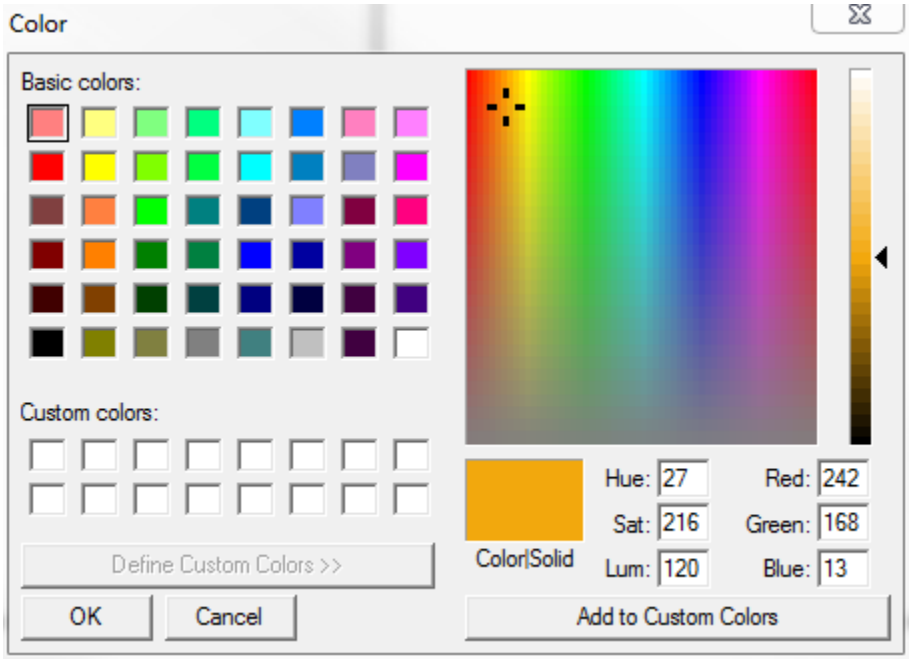



Fig. 8: The color scheme window that enables the users to choose the beginning and the ending colors for the animation.

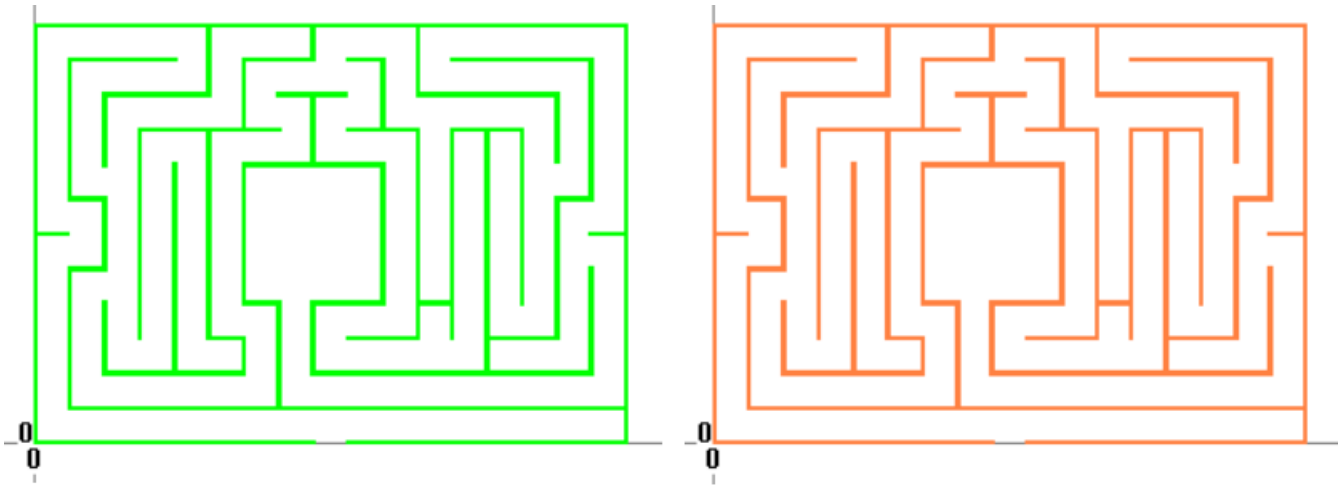


Fig. 9: Maze color changes from green (on the left) to orange (on the right).

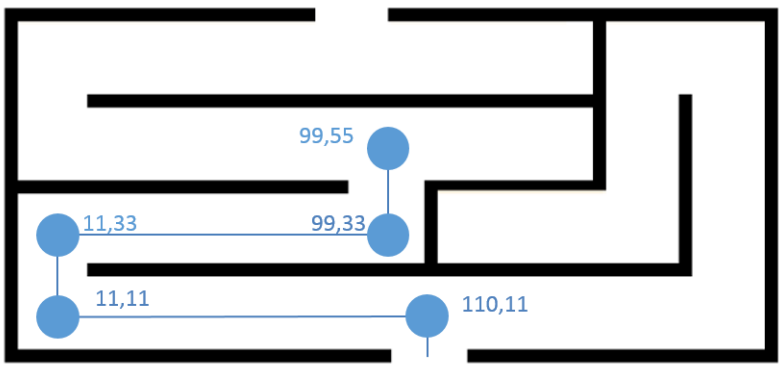



Fig. 10: A moving object within a maze.

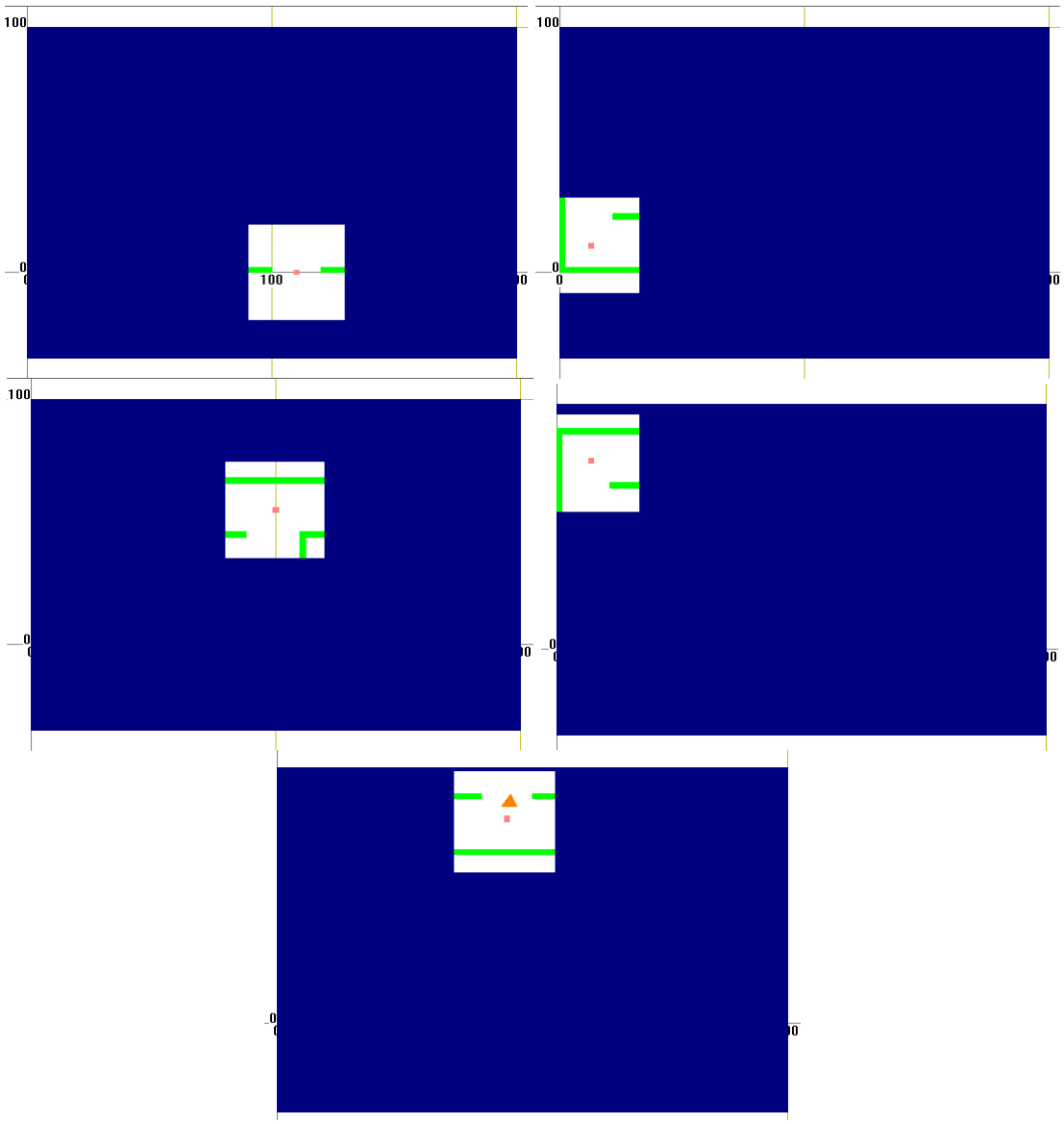
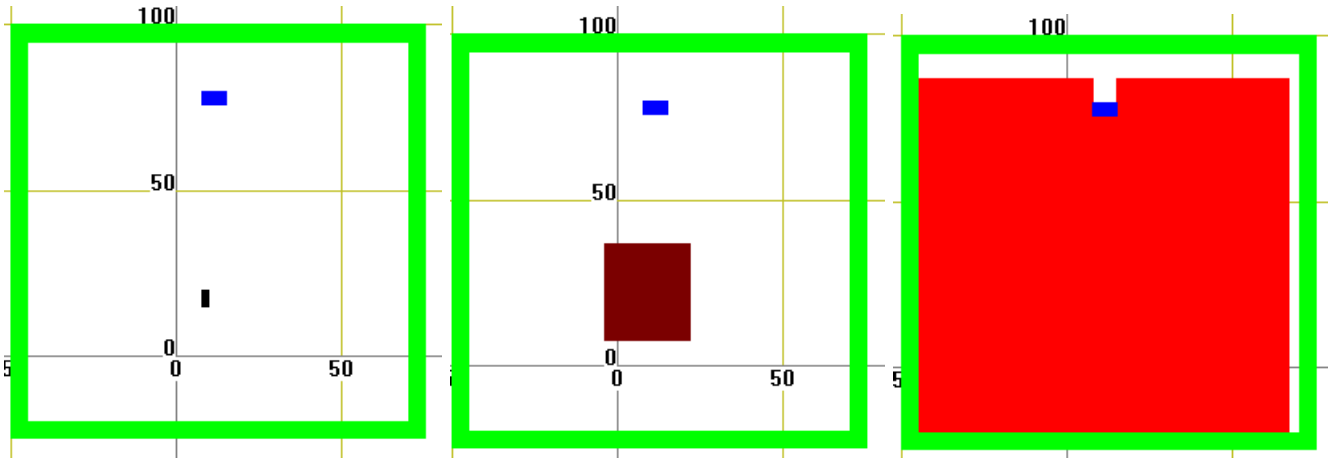
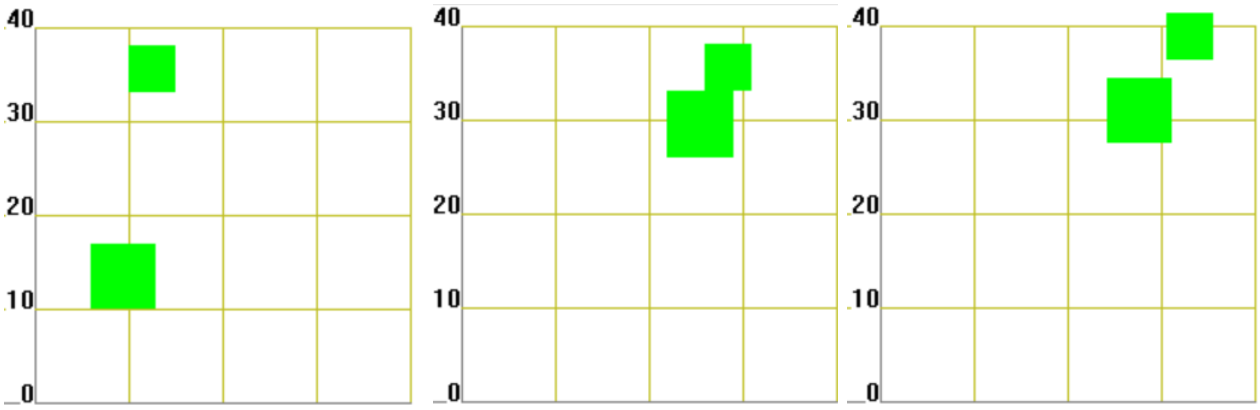


Fig. 11: An example of the use of a searchlight for games. The searchlight allows the players to see only a small area of the entire maze, making exploration of the maze much harder.




(a) (b) (c)

Fig. 12: An exploding grenade with the explosion slowly turning from black to red and being blocked from expanding further by a green wall and a blue column. Above are only three snapshots (a)-(c) from an animation of the explosion.




(a) (b) (c)

Fig. 13: Three snapshots of an animation of two objects colliding and bouncing back from each other.

### B. Collision of Objects

Another interesting function is called collide where one has to specify an attribute called mass that when the two objects come in contact with each other will apply a collision that will cause the elastic objects to bounce back from each other. This operator works well for car crashes, for example. Collision is different from the block operator. The block operator only stops the object from moving past the other object, but with the collision function the bouncing back from each other can be also represented and animated, enabling the design of more interesting games.

## VI. Conclusion and Future Work

Constraint databases and video games have been around for a long time, but they have never been combined together before. The A-Maze-D system shows that constraint databases can be applied conveniently and efficient to design of maze games. A-Maze-D provides a versatile set of features by a combination of a MATLAB library and the MLPQ constraint database system. This is the first time that maze games have been created using constraint databases.

In the future we would like to try implement different game types in constraint databases and see how well they can be converted over from the normal game development into constraint databases. We believe the best way to do this is testing out different types of games and see how well they perform. A few examples to name would be to test games that are more like Tetris where one needs a faster reaction or one's actions are limited by the time allotted.

In addition to interactive games, the A-Maze-D system could be also used for the simulation of interactive control processes, such as the simulation of the control processes in coal mining [16]. The simulation of coal mining is an interesting application because it requires 3-D spatial representation and animation, which can be also provided in constraint database systems like MLPQ and added to the A-Maze-D system [21].

Examining new possible features that constraint databases can provide for game development is another area we would like to research more and test out. Constraint databases have been used before in the animation of human faces. Hence an intriguing possibility is to allow players to speak to each other. Whenever a player wants to say something a pop-up window would open and show an animation of the player's face as he or she speaks.

## References

[1] A. Anderson, P. Z. Revesz, Efficient MaxCount and threshold operators of moving objects, *Geoinformatica*, 13 (4), 2009, pp. 355–396.

[2] R. L. Burden, J. D. Faires, *Numerical Analysis*, 9th edition, Springer, New York, USA, 2014.

[3] J. Chai, *Exploiting Spatial-Temporal Constraints for Interactive Animation Control*, Ph.D. Dissertation, Carnegie Mellon University, Pittsburgh, 2006.

[4] J. Chomicki, P. Z. Revesz, Constraint-based interoperability of spatiotemporal databases, *Geoinformatica*, 3 (3), 1999, pp. 211–243.

[5] K. Claypool, M. Claypool, *Teaching software engineering through game design*, *ACM SIGCSE Bulletin*, 37 (3), 2005.

[6] N. Cressie, C. K. Wikle, *Statistics for spatio-temporal data.* John Wiley & Sons, 2011.

[7] D. H. Eberly, *Game physics.* Taylor and Francis, 2010.

[8] A. Herwig, P. Paar. Game engines: Tools for landscape visualization and planning. In: *Trends in GIS and Virtualization in Environmental Planning and Design*, E. Buhmann, editor, Wichmann Publishing, 2002, pp. 161-172.

[9] J. Komarkova, et al., Methods of usability evaluation of Web-based geographic information systems. *International Journal of Systems Applications*, Engineering & Development 5 (1), 2011, pp. 33-41.

[10] P. C. Kanellakis, G. M. Kuper and P. Z. Revesz, Constraint query languages, *Journal of Computer and System Sciences*, 51 (1), pp. 26-52, 1995.

[11] H.-P. Kriegel, M. Schubert, A. Zfle, Managing and mining multiplayer online games, *Proc. of the 12th International Conference on Advances in Spatial and Temporal Databases*, Springer Berlin, 2011, pp. 441-444.

[12] M. Lewis, J. Jacobson, Game engines, *Communications of the ACM*, 45 (1), 2002, pp. 27.

[13] L. Li, and P. Z. Revesz, Interpolation methods for spatio-temporal geographic data, *Computers, Environment and Urban Systems*, 28 (3), 2004, pp. 201–227.

[14] W. H. Matthews, *Mazes and labyrinths: their history and development*, Courier Corporation, 1970.

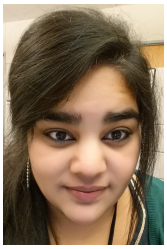
[15] I. Millington, *Game physics engine development*, Amsterdam: Morgan Kaufmann Publishers, 2007.

[16] V. Okolnishnikov, S. Rudometov, S. Zhuravlev, Simulation environment for development of automated process control system in coal mining, *Proc. International Conference on Systems, Control, Signal Processing and Informatics*, 2013.

[17] F. A. C. Perrin, An experimental and introspective study of the human learning process in the maze, *Psychological Monographs: General and Applied*, 16 (4), 1914, pp. 1-97.

[18] P. Z. Revesz, *Introduction to Databases: From Biological to Spatio-Temporal*, New York, USA: Springer, 2010.

[19] P. Z. Revesz, A recurrence equation-based solution for the cubic spline interpolation problem, *International Journal of Mathematical Models and Methods in Applied Sciences*, 9, 2015, pp. 446 - 452.
[20] P. Z. Revesz, C. Assi, Data mining the functional characterizations of proteins to predict their cancer-relatedness, *International Journal of Biology and Biomedical Engineering*, 7 (1), 2013, pp. 7–14.
[21] P. Z. Revesz, R. Chen, P. Kanjamala, Y. Li, Y. Liu and Y. Wang, The MLPQ/GIS constraint database system, *ACM SIGMOD International Conference on Management of Data*, ACM Press, 2000.
[22] P. Z. Revesz, T. Triplet, Classification integration and reclassification using constraint databases, *Artificial Intelligence in Medicine*, 49 (2), 2010, pp. 79–91.
[23] P. Z. Revesz, S. Wu, Spatiotemporal reasoning about epidemiological data, *Artificial Intelligence in Medicine*, 38 (2), 2006, pp. 157-170.
[24] S. Rogers, *Level Up! The guide to great video game design*, John Wiley & Sons, 2014.
[25] W. S. Small, Experimental study of the mental processes of the rat. II *The American Journal of Psychology*, 1901.
[26] J. B. Watson, Kinaesthetic and organic sensations: Their role in the reactions of the white rat to the maze. *The Psychological Review: Monograph Supplements*, 8 (2), 1907.



**Shruti Daggumati got her M.S. in Computer Science from the University of Nebraska-Lincoln in May 2015. Her primary research area is data visualization and animation with a focus on data mining and visual analytics of large temporal data sets and finding correlations among independent attributes.**

**Peter Z. Revesz holds a Ph.D. degree in Computer Science from Brown University. He was a postdoctoral fellow at the University of Toronto before joining the University of Nebraska-Lincoln, where he is a professor in the Department of Computer Science and Engineering. Dr. Revesz is an expert in databases, data mining, big data analytics and bioinformatics. He is the author of Introduction to Databases: From Biological to Spatio-Temporal (Springer, 2010) and Introduction to Constraint Databases (Springer, 2002). Dr. Revesz held visiting appointments at the IBM T. J. Watson Research Center, INRIA, the Max Planck Institute for Computer Science, the University of Athens, the University of Hasselt, the U.S. Air Force Office of Scientific Research and the U.S. Department of State. He is a recipient of an AAAS Science and Technology Policy Fellowship, a J. William Fulbright Scholarship, an Alexander von Humboldt Research Fellowship, a Jefferson Science Fellowship, a National Science Foundation CAREER award, and a Faculty International Scholar of the Year award by Phi Beta Delta, the Honor Society for International Scholars.**

**Corey Svehla is currently pursuing his masters degree in Computer Science at the University of Nebraska- Lincoln. He particularly enjoys finding new ways to integrate technology with agriculture to make farmer's lifestyle easier.**