# Applying generic programming algorithms to act as an artificial intelligence

Jacob Johnson

June 3, 2004

**Abstract**

To date, most artificial intelligences are simple programs that are unable to 'learn' (in the sense that they cannot rewrite themselves). These AIs range from simple search algorithms to multi-faceted human imitation programs. The purpose of this project is to show a way that one of these programs can work. The bulk of the project uses simple math and physics equations, so most of the research is focused on the use of OpenGL.

## 1 Introduction

This project is composed of a research area and a project area. My research centers around the manipulation of OpenGL to create a roughly humanoid character model with moving body parts, including two arms, a head, half of two legs, and a torso. The project area involves creating an algorithm that allows the character to respond to (preset) changes in his environment. Hopefully, the combination of the two will give the program a realistic feeling.

## 2 Background

I am using a program which I began writing last year for Supercomputing Applications (dodge.cpp). I salvaged the program through and old web site and was able to continue to work on it. As a result, I had most of the supplementary work for the project done, and had only to install the part relating to the project. The creation of the algorithm itself was the most interesting part. With random input entered in accordance with a given set of parameters, a series of functions generates a response which is sent back to my main program and applied to a humanoid model, which acts a certain way according to its input. The response and the input given work together to give the model the visual effect of possessing intelligence. The program is fully capable of ”dodging bullets” at this point in time. Unfortunately, because of a last-minute change to the way these computers run OpenGL applications, the program no longer displays itself correctly. Since the change was made in the latter half of the fourth quarter, I had no time to even try to fix it. The program once was visually pleasing, as the work I have included here is part of a larger-scale project that relied on aesthetic quality.

## 3 OpenGL

I'm using a texture tutorial to learn how to texture my program. The end result should be a more realistic environment, hopefully with texture-drawn clouds instead of the current 3D ones, and textured buildings. The lighting system, although not up to par with what I would hope for, effectively accomplishes what it needs to (namely, lighting the scene). The sun should to evenly light everything in the scene (although without shadows), and other various lights like muzzle flare should drop off, but with OpenGL it is a bit hard to do so. Other than that, all OpenGL works were made last year in Supercomputing Applications using simple geometry like spheres, prisms, and tori.

## 4 Emergent Behavior

The research area with which I'm working is emergent behavior. In my project specifically, emergent behavior will exist in a basic form with the compound movements involved in the character's 'dodging' ability. If his arm is moving to avoid a specific projectile, it could be aided by the movement of his torso (caused by another projectile). This is the simplest form of emergent behavior, but is an understandable study of how it works. Emergent behavior is most commonly associated with the social behavior of insects. The most regularly observed example of EB is the insect swarm. While swarming, insects will exhibit behavior not observed in individual insect actions. The collective mind of the whole will make all of the smaller elements act as a larger entity, using mechanisms simple enough to exist in some of the most basic and earliest forms of life on earth. I have been doing research outside of what

I need specifically for my project. I've learned of several examples of emergent behavior being used in computer programming. On of the more prominent texts is a novel by Michael Crichton entitled Prey. Prey is the story of nanobots with the capability of exhibiting EB attacking scientists in a laboratory in the middle of the desert. The novel goes into many aspects of the capability of these creatures, as is a very good example of how emergent behavior and computer science can interact.

## 5 Program Layout

The algorithm that will be the center of the program will use algebra to derive the movement of incoming projectiles, and will act accordingly. If one of the projectiles is headed towards the character, he will respond appropriately by attempting to move out of the way. If he is unable to avoid the projectile with any movement possible, he will be hit and the program will visually display it and mark it in text. At this point in the evolution of the program, the character can move his entire body in accordance to the projectiles' movements, but the body doesn't work together. Although the torso will bend back, moving the arms with it, the program still thinks the arms are where they originally started, meaning the program moves them to dodge projectiles not fired at them.

## 6 Primary Function

The algorithm itself has a two-fold function. First, it analyzes the movement of any incoming projectiles to find their x, y, and z velocity vectors. Then, has to determine where and when the projectile would come in contact with the character, determining the point of evasion. After that, the program has to find a safe position for the troubled body part elsewhere in the character's motion range. After all this, the second part of the program kicks in. It has to determine the most efficient way to rotate the limbs and torso to move the body to the designated position. The function shown here is the "dodge" function, which takes parameters from the projectiles and the limbs and determines if they come in contact, and if so, what to do about it.

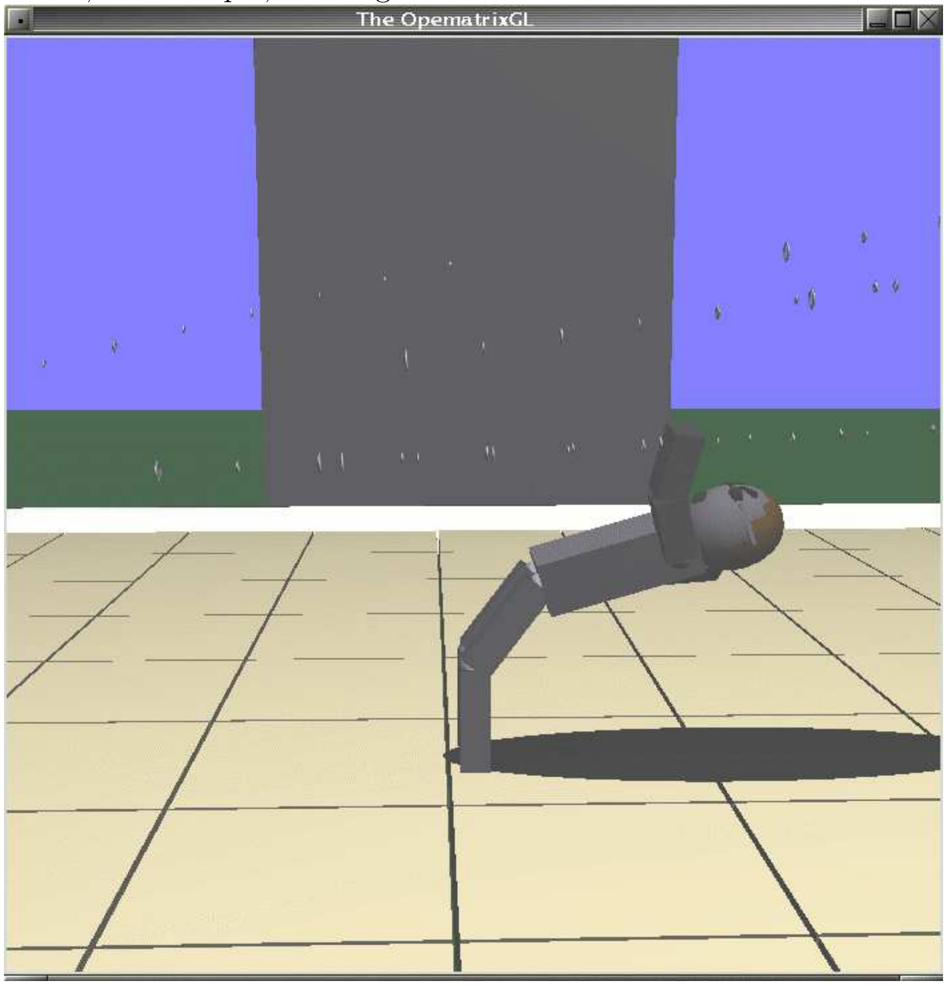
```
if( (incs*deltax[nn] - centx)*(incs*deltax[nn] - centx) + (incs*deltay[nn] - centy)
        *(incs*deltay[nn] - centy) <= radius && bullz[nn] > 0.0)
{
if(getdod[nn] == 0)
{
getdod[nn] = -1;
if(p_part < -45)
getdod[nn] = 1;
}
ppart = 2*getdod[nn]*(15 - (int) floor(bullz[nn]));
if(p_part <= llim)
getdod[nn] = 1;

if(p_part >= ulim)

getdod[nn] = -1;

}
```

MMMKThe variable 'nn' here is used instead of the traditional 'n' becuase of simoltaneous processes running and using the global 'n' variable. The variables p_part and ppart represent the rotation and identification of a limb.

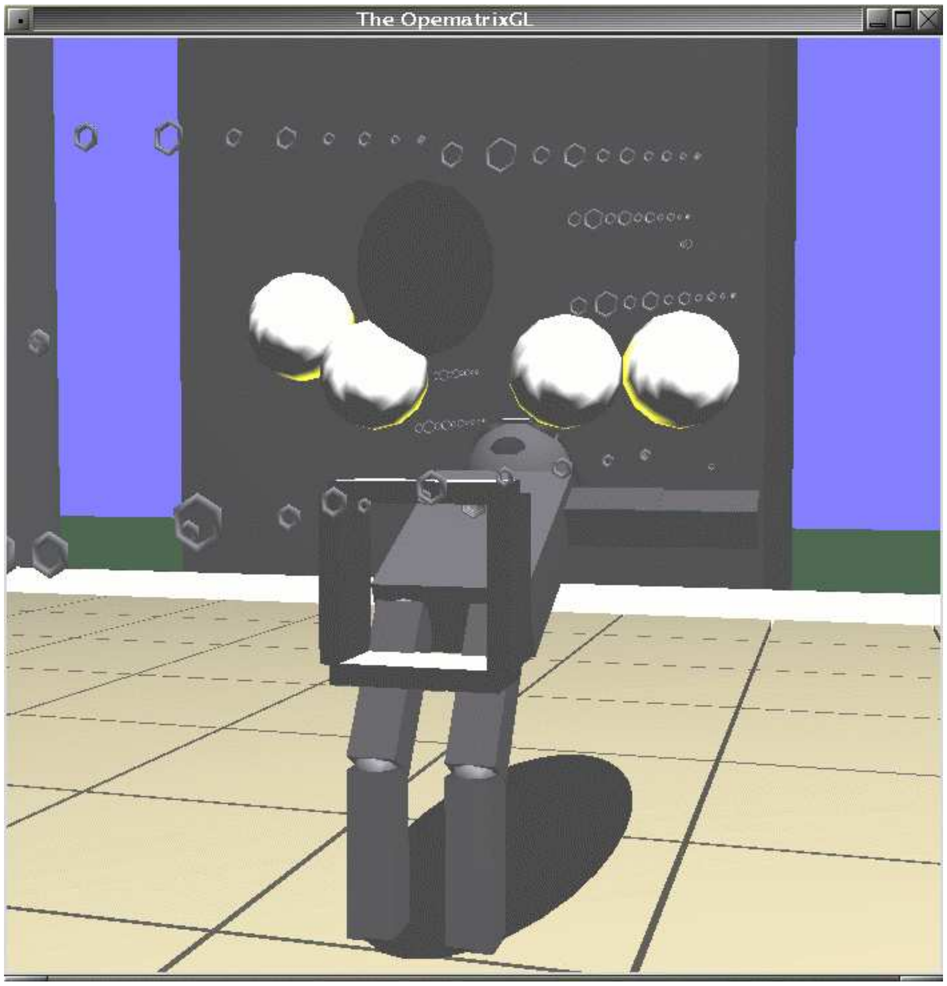
The variable 'incs' refers to the number of frames the projectile will travel to get from the firing point to the collision point, and is determined based on a delta-z value. This is multiplied by 'delta{$x$, y}' to find the x and y coordinates where the projectile will pass the plane of the character. From there, a simple collision detection algorithm sees if the arm is within range of that shot. Unfortunately, as it is now, the program detects whether the arms will get hit by replacing them (in the computer's mind) with a sphere and seeing if the point passes through the sphere. This is much easier and faster than a normal collision detection algorithm, but it unfortunately lacks the accuracy I would normally have. If all the conditions are met, the program continues to move the arm accordingly.

After each of the projectiles cycles through this algorithm to determine if a problem exists, the program starts over again with a new frame. During the in-between time, the body and the projectiles will move, meaning that the program will have to re-run the algorithm each time.

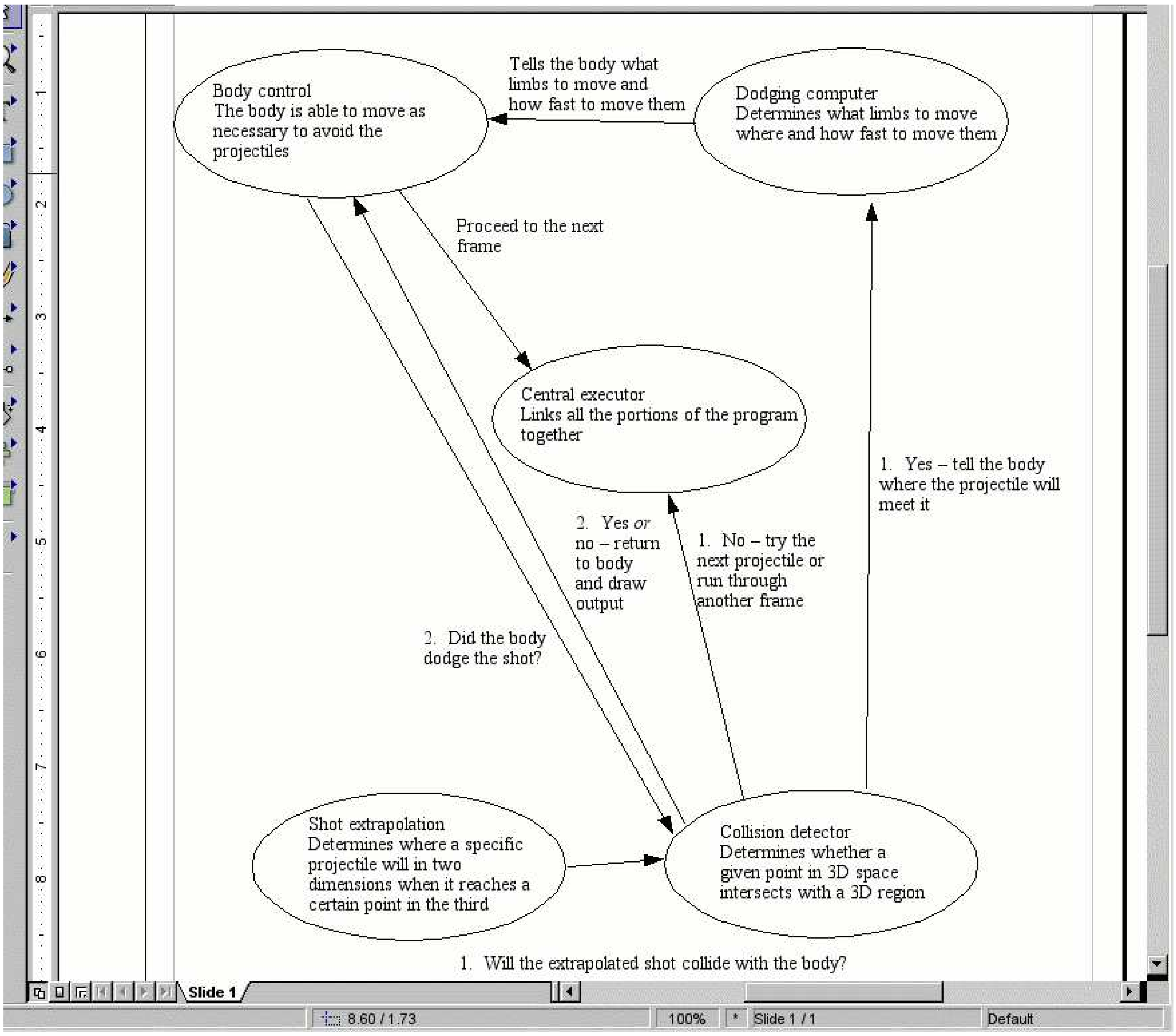
The variable 'getdod' is tells the program if the projectile is being dodged by the limb, and if so, in what direction. The direction is determined in conjunction with the 'llim' and 'ulim', which specify the rotational limits of a limb; this basically means that it prevents an arm from, for example, bending backwards at the elbow.



`A shot of the character from the side, bent over.`



A shot of the character bending over, showing the various shapes used to detect collisions. At this point the head collision sphere doesn't work, so it is (naturally) in the wrong place.


Body control
The body is able to move as necessary to avoid the projectiles
Tells the body what limbs to move and how fast to move them
Dodging computer
Determines what limbs to move where and how fast to move them
Proceed to the next frame
Central executor
Links all the portions of the program together
1. Yes – tell the body where the projectile will meet it
2. Yes *or* no – return to body and draw output
1. No – try the next projectile or run through another frame
2. Did the body dodge the shot?
Shot extrapolation
Determines where a specific projectile will in two dimensions when it reaches a certain point in the third
Collision detector
Determines whether a given point in 3D space intersects with a 3D region
1. Will the extrapolated shot collide with the body?
Slide 1
8.60 / 1.73
100%
Slide 1 / 1
Default

A flowchart I made of the program’s operation. Zoom in to read it.

## 7 Works Cited

Nitschke, Geoff. Emergent Specialized Behaviour in a Pursuit-Evasion Game. Jan 21, 2004. <http://www.ifi.unizh.ch/ailab/people/nitschke/Abstracts/evoc-abstract2.html>

Kitano, H., Asada, M., Noda, I., and Veloso, M. (1999). RoboCup: Today and Tomorrow - What we have learned. Artificial Intelligence, vol. 110(2): 193-214. Elsevier Science Publishers, Amsterdam, Holland.

Varner, Philip E. and Knight, John C. Jan 14, 2002. Monitoring and Visualization of Emergent Behavior in Large Scale Intrusion Tolerant Distributed Systems. Jan 21, 2004. <www.cs.virginia.edu/~pev5b/research/papers/wits-dsn2002.pdf>