Practical Applications of Botball Through Video Games
Marty Rand
Norman Advanced Robotics
marsedge95@gmail.com

Practical Applications of Botball Through Video Games

Introduction

Botball prides itself on teaching real-world, practical engineering skills. As video game development involves a mix of programming, engineering, and documentation, Botball also teaches many concepts needed for game development. This paper details several of these lessons that Botball teaches.

Game Background

The game made using these concepts is called Project Land Mineded[i] (PLM). PLM is an arena-based first-person shooter where bullets ricochet off walls (using geometrically accurate projectile physics), bounce to the ground, and become land mines. The entire battlefield gets covered with mines that stay there until someone steps on one. The game doesn't take itself too seriously and instead focuses on having fun. Players can adjust the physics engine to be more realistic or more over the top by changing gravity and the collision elasticity.

Programming Connection

PLM is made using Unreal Engine 3 (UE3)[ii], written in C++, with added scripts written in UnrealScript[iii]. UnrealScript is a proprietary language made specifically for Unreal Engine. UnrealScript is based on Java, which is based on C++, which is based on C – the language Botball uses most. Even though this connection is distant, Botball programmers can understand some UnrealScript code. For instance: (see right). This code is from the AI of the game. The code is simply a couple of if/else blocks of code – something many Botballers use every year. Both C++ and UnrealScript are object-oriented. Video game development is very suited for object-oriented languages because of the different game objects in the scene. Botball has added C++ support in recent controller versions, keeping up-to-date with the changing field of Computer Science.

```
if ( CanCamp() ){ //if can stay in same spot
    GoalString = "Camping";
    GotoState('Camp');
    return;
}
else if ( LoseEnemy() )
{//else if can't find anyone
    WhatToDoNext();//figure out what to do
    return;
}
else
{//else go back where it was
    GoalString = "Stop Looking";
    DoRetreat();
    return;
}
```

The KISS Principle

The KISS principle (Keep it Simple Stupid) is so fundamental to Botball that it is included in the acronym for KIPR – The KISS Institute for Practical Robotics[iv]. KISS is used for all kinds of engineering. For instance, no competent programmer uses recursion instead of a simple for loop in C. A robotics example would be ramming a wall until the robot is straight instead of using a touch sensor to detect when the robot hits the wall. Similarly, PLM uses timers instead of sensors to save CPU cycles and thereby optimize the code. PLM does this when a player spawns into the level. PLM applies a custom material to the player model. The part of code dealing with applying the material has no knowledge of when the player actually spawned. It just uses a small timer to approximate it. It may be a slightly worse practice, but it works just fine and there is no reason to change it. This has the pleasant side effect of being able to ignore parts of the engine and work around them by using these timers.

Iterators

Botball camera code often iterates through the first n camera blobs to find the appropriate one and then performs some kind of operation using that blob. Just as cameras have an arbitrary number of blobs to check, PLM iterates through all PlayerController instances to find the local player, and then allows that player to control the menu.

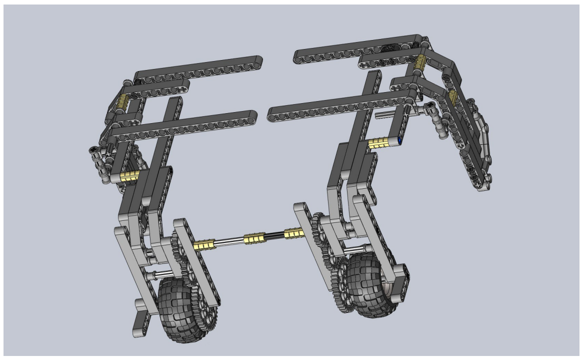
```
//PLM
foreach WorldInfo.AllControllers(class'MarsPlayerController', Contr)
{//iterate through all players
        Contr.GetPlayerViewPoint(AimLocation, AimRotation);
        //get direction player is aiming
        AimDirection=vector(AimRotation);
        //calculate of player is looking at a button
        SeenActor = Trace(HitLocation, HitNormal, AimLocation + AimDirection * MaxDistance, AimLocation, true);
        //if player is looking at a button
        if(SeenActor != None && SeenActor.IsA('Button_Key'))
        {
                MarsGameReplicationInfo(WorldInfo.GRI).ActiveKey = Button_Key(SeenActor);//activate the button
                return true;
        }
}

//Botball
while(found==0 && seconds() < (timeInit+timeout) )//while no object found and within time limit
{
        depth_update();
        depth_scanline_update(row);
        if(get_depth_scanline_object_count() > 0)//if objects exist
        {
                for(objCount=0;objCount<get_depth_scanline_object_count()&&found==0;objCount++)
                {//iterate through all objects looking for the correct kind
                        if(get_depth_scanline_object_center_x(objCount) > (column - error) &&
                                get_depth_scanline_object_center_x(objCount) < (column + error) &&
                                get_depth_scanline_object_center_z(objCount) < maxDepth)
                        {//if the correct kind is found, break out of loop
                                found=1;
                                printf("found\n");
                        }
                }
        }
        msleep(15);
}
```

Debugging

One of the most important parts of programming is debugging. Botball mainly uses printf to display information on the Link's screen. Since a game is entirely virtual, UE3 has a debug window that displays various information about run-time errors, what the engine is doing, and user debug. In UE3, the command `log("..."); displays "…" (without the quote marks). UE3 also has the ability to overlay arbitrary information onto the Heads-Up-Display (HUD). This allows the programmer to watch information as the game plays out without having to look at a dedicated debug window. UE3 can also display any object's name in the scene. This is useful for figuring out if there is a duplicate, if the program is accessing an object that doesn't exist, etc. This ability exceeds the Link's because Botball engineers often make the Link's screen difficult to clearly see while the robot it running – using it as a counter-weight, hiding it under a basket, making a giant arm raise and lower in the way, etc.

CAD/3D Modeling

Some of the more experienced engineers build their robots in SolidWorks[v] before using real Legos. This lets them prototype different design strategies without having to dig up all the needed Legos. PLM uses 3D models for all geometry in the game. The process is similar between CAD modeling and 3D modeling. In some ways, SolidWorks is better than Blender[vi] (the 3D modeling tool I use). SolidWorks makes it easy to "drill" a hole into a mesh. Blender does not make that as intuitive.



Project Documentation

All Botball teams must keep detailed documentation of what they have done, failed ideas, etc. This is a good practice in any large project – including game development. A common theme is to have a design document that details everything the game will do. In other words, documentation for the game.

Documenting Code

One of the first things to skip when running out of time is documenting code. Those pesky comments eat into a team's precious last week before competition. Still, documentation is critical to having effective and reusable code. UE3 was not made by me, nor do I have full source code. As such, its creators must document it well enough for users to understand what is going on. This took the creators a large amount of time, but it was necessary. Real-world programming projects need proper documentation to survive the new intern and disgruntled client. Documentation is critical to having fully functional code. Botball is a great way to get started in that area of programming.

Conclusion

While it is impossible to list every advantage Botball gives for game development, I hope this list gives a useful introduction to the similarity in skill sets.

i Martion Laboratories – http://www.martionlabs.com/
ii Unreal Development Kit – https://www.unrealengine.com/products/udk/
iii UnrealScript – https://udn.epicgames.com/Three/UnrealScriptHome.html
iv KIPR – http://www.kipr.org/
v SolidWorks – https://www.solidworks.com/
vi Blender – http://www.blender.org/