```
define crash-and-turn
    turn parallel to wall
    while (not done)
        if (wall_side_clear)
            turn to wall side
            move_forward
            done = true
        else if (front clear)
            move_forward
        else
           crash-and-turn
        end if
    end while
end crash-and-turn
```

the WorldModel can give you.

Lists and iterators

We have provided you with a list class and an iterator for traversing the list. Of course, you are free to use your own list classes if you so choose, but ours is the one which the World-Model uses, so you have to use it too. Ha!

The list, iterator, and associated comments can be found in HashListT.h and HashListIterator.h. There is some sample code which demonstrates its use in the *grabFlag* example tank. If you're not satisfied with that example, you can find further examples scattered about in the support code for the client and the server. Have fun.

Shortest Path Algorithm

There are a large number of planning algorithms, from the very simple "crash-and-turn" algorithm to the more complex A* algorithm. We'll explain the crash-and-turn algorithm here, as it is fairly simple and works well for the base requirements.

The algorithm is very similar to its name. The algorithm works so that you move from one spot to a spot closer to your goal. If this path is blocked, then it follows the wall around until it has maneuvered around the object, and is able to go towards the goal. This is very simple to implement in our grid world.

Pseudo-code for Crash-and-Turn

```
define move-to-goal
     if (front is toward goal)
          if (front is clear)
               move_forward
          else
               crash_and_turn
          end if
     end if
end move-to-goal
```

Coding a tank

All that you need to do is implement a class which inherits from the abstract AI class. The interface for the AI class is found in AI.h.

Actions

The AI class provides a set of actions which allow your tank to operate in the world:

- *shoot*() shoots a bullet in the direction that the tank is facing.
- *move_forward*() moves one space in the tank's current direction.
- *move_backward*() moves one space opposite to the tank's current direction.
- *turn_right*() rotates the tank 90 degrees to the right.
- *turn_left*() rotates the tank 90 degrees to the left.
- *communicate*() sends a message to all of the other tanks.

Making moves

The tank is set up so that it gets the moves in the function *getMove*(). This is when you must return a move to the client. Think as long as you want, but beware that if you take too long, the game will assume that you don't want to make a move that turn!

Events

The tanks are given callbacks by the server to tell it when certain events have occurred. The callbacks are implemented as virtual functions in the AI class. You can redefine these methods to do anything you want or you can simply ignore them. Examples of some callbacks that you can receive are:

- *wasHit*() - called if another tank has shot you; it will tell you who the shooter was.
- *pickedUp*() - called if your tank picks up an item, such as ammo or the flag.
- *gotMessage*() - called when your tank intercepts a message from another tank.

Info

You have various functions to get your location, home's location, hit-points, amount of ammo you have, etc. You can find these functions commented in AI.h.

The WorldModel

Your tank is in a world with a bunch of other tanks. This world is constantly changing, as tanks move about and shoot one another. The *WorldModel* class maintains an image of the current state of the world. There is a global variable defined in TankGlobals.h called *G_world*, which contains the current state of the world. See WorldModel.h to find out what kind of information

placed on the field. To keep you players supplied while you are away from your base, packets of ammunition, fuel, and armor are randomly placed around the field. You obtain these supplies by simply running over them. They will increase your armor, ammunition, or fuel supplies by a certain amount. You can win the game in one of two ways: either survive, while the rest of the tanks are killed off; or you can be the first person to bring five flags back to your home base.

Grading Requirements

Now, as this is actually for a class, we must put some requirements on what your tank must do. We debated grading criteria for a while. After deciding that "just make it cool" was not really a specific enough criteria, we have decided on the base spec:

You must have one tank that implements a path planning algorithm to enable it to get to the flag and back to its base in a world with no enemies. However, feel free to make as many tanks as you want. These tanks can implement any number of strategies, from the "kill everything that moves" strategy to the "hide in the corner until it's over" strategy; or you can program a whole fleet of AI tanks to implement a team strategy. The possibilities really are endless, and frankly, we really feel, you should drop all your other work in an effort to make the best possible tank. At the end, there will be a free-for-all competition between all of the tanks which you create. We encourage you to implement interesting and unusual behaviors for your tank.

Support Code

The support code is located in /course/gs007/asgn/ai. There are example tanks in /course/gs007/asgn/ai/examples. This directory also holds the Makefile you will need to make your tanks. Copy it from this directory to your working directory. All of the relevant header files are located in /course/gs007/asgn/ai/include.

Example tanks

Look at the examples in the examples directory (you can even copy the code!):

- *sampleTank* is the basic framework for your tank, use it as a template for your tank files. The other examples give some examples of the commands that the tank has available.
- *loudTank* shows how a tank can communicate with other tanks.
- *playerTank* is actually a hack which allows you to play along with the AI tanks!

ThinkTanks
Fun with Brains

Is your life filled with trying exams or overly time consuming cs32 projects? Well, my friends, we have the ultimate escape for you. Play ThinkTanks and have all the fun of storming across the desert trying to get the last precious flag back to your base, while your friends stalk you down intent on killing you for the precious flag, without all the noise, dust, or actual death and carnage. Best of all, with ThinkTanks you don't even have to deal with annoying keystrokes. Just program up your favorite strategy, then you're free to mindlessly watch your tank be pummeled by the tanks of the AI group.

Now, dear reader, you're probably thinking this sounds to good to be true, and since you have plenty of free time, do you really need that four hours of sleep? So, why not play ThinkTanks? First, you think, maybe I ought to find out more about this game. Luckily we have more information for you.

The Game

As the name implies, ThinkTanks does involve tanks. You, the user or your AI stunt double, are in control of a tank that competes with other tanks in what amounts to a big ol' game of capture the flag. Your tank starts off on its home base, which is placed at a random location on the field of play. When you are on your home base, you are not protected from the enemy. However, when you remain on your home base, your armor, fuel, and ammunition supplies will gradually increase. When you venture out into the real world, your main objective will be to collect the flag, that will be randomly placed around the field, and deposit it back to your home. However, you must also avoid being shot by competing players who will try to kill you either to eliminate you as competition or to keep you from returning to your home with the flag. Once a player has the flag, it can only be brought back into play by killing the player before that player reaches their home base. Once the flag carrier is killed, the flag is placed on the location of destruction, and is fair game. To protect yourself from these avenging hordes of tanks, you can hide behind the walls