Agent with Prediction & Visual Debug

A key capability that had been missing from Geometry Friends' interface for AI agents is the prediction of actions. In reality this is the ability to predict the game state at a given time ahead in the future. Additionally, another barrier to more detailed and efficient debugging of AI agents implemented was the lack of a visual debugging mechanism that could be used directly from the agent implementations.

Both the described capabilities are offered starting Geometry Friends Competition 2016. The purpose of this article is to introduce how to use these capabilities. As a starting point for this code example we use the Example: The Random Circle Agent. We then describe the modifications needed to demonstrate the newly introduced capabilities by setting the following requirements for our Corcle Agent:

1. Given the current state of the circle and its current action:
    1. Be able to predict the circle path 2 seconds ahead in the future;
    2. Be able to predict if the agent will catch any collectible in the following 2 seconds;
2. Be able to keep track of the collectibles that have been caught and mark them visually in the game area;

Remember that to see this visual debug information you have to press the F1 key during the game.

Download complete source file here.

Additional Variables Declared

```
//predictor of actions for the circle
private ActionSimulator predictor = null;
private DebugInformation[] debugInfo = null;
private int debugCircleSize = 20;

//debug agent predictions and history keeping
private List<CollectibleRepresentation> caughtCollectibles;
private List<CollectibleRepresentation> uncaughtCollectibles;
private object remainingInfoLock = new Object();
private List<CollectibleRepresentation> remaining;
```

Additional Constructor Initializations

```
        //history keeping
        uncaughtCollectibles = new List<CollectibleRepresentation>();
        caughtCollectibles = new List<CollectibleRepresentation>();
        remaining = new List<CollectibleRepresentation>();
```

Additional Setup Processing

```
        uncaughtCollectibles = new List<CollectibleRepresentation>(collectiblesInfo);
```

Additional SensorsUpdated Handling

```
       /*WARNING: this method is called independently from the agent update - Update(TimeSpan elapsedGame
       * structures that are modified in both (e.g. see operation on the "remaining" collection)
       */

        lock (remaining)
        {
            remaining = new List<CollectibleRepresentation>(collectiblesInfo);
        }
```

Overriding ActionSimulatorUpdated Handling

```
//provides the circle agent with a simulator to make predictions about the future level state
public override void ActionSimulatorUpdated(ActionSimulator updatedSimulator)
{
    predictor = updatedSimulator;
}
```

Additional Update Processing

```
        ...
        //check if any collectible was caught
        lock (remaining)
        {
            if (remaining.Count > 0)
```

```
    {
        List<CollectibleRepresentation> toRemove = new List<CollectibleRepresentation>();
        foreach (CollectibleRepresentation item in uncaughtCollectibles)
        {
            if (!remaining.Contains(item))
            {
                caughtCollectibles.Add(item);
                toRemove.Add(item);
            }
        }
        foreach (CollectibleRepresentation item in toRemove)
        {
            uncaughtCollectibles.Remove(item);
        }
    }
}

//predict what will happen to the agent given the current state and current action
if (predictor != null) //predictions are only possible where the agents manager provided
{
    /*
     * 1) simulator can only be properly used when the Circle and Rectangle characters are ready,
     * 2) in this implementation we only wish to simulate a future state when whe have a fresh si
    */
    if (predictor.CharactersReady() && predictor.SimulationHistoryDebugInformation.Count == 0)
    {
        List<CollectibleRepresentation> simCaughtCollectibles = new List<CollectibleRepresentatio
        //keep a local reference to the simulator so that it can be updated even whilst we are pe
        ActionSimulator toSim = predictor;

        //prepare the desired debug information (to observe this information during the game pres
        toSim.DebugInfo = true;
        //you can also select the type of debug information generated by the simulator to circle
        //toSim.DebugInfoSelected = ActionSimulator.DebugInfoMode.Circle;

        //setup the current circle action in the simulator
        toSim.AddInstruction(currentAction);

        //register collectibles that are caught during simulation
        toSim.SimulatorCollectedEvent += delegate(Object o, CollectibleRepresentation col) { simC

        //simulate 2 seconds (predict what will happen 2 seconds ahead)
        toSim.Update(2);

        //prepare all the debug information to be passed to the agents manager
        List<DebugInformation> newDebugInfo = new List<DebugInformation>();
        //clear any previously passed debug information (information passed to the manager is cum
        newDebugInfo.Add(DebugInformationFactory.CreateClearDebugInfo());
        //add all the simulator generated debug information about circle/rectangle predicted path
        newDebugInfo.AddRange(toSim.SimulationHistoryDebugInformation);
        //create additional debug information to visualize collectibles that have been predicted
        foreach (CollectibleRepresentation item in simCaughtCollectibles)
        {
            newDebugInfo.Add(DebugInformationFactory.CreateCircleDebugInfo(new PointF(item.X - de
```

```
                newDebugInfo.Add(DebugInformationFactory.CreateTextDebugInfo(new PointF(item.X, item.
            }
            //create additional debug information to visualize collectibles that have already been ca
            foreach (CollectibleRepresentation item in caughtCollectibles)
            {
                newDebugInfo.Add(DebugInformationFactory.CreateCircleDebugInfo(new PointF(item.X - de
            }
            //set all the debug information to be read by the agents manager
            debugInfo = newDebugInfo.ToArray();
        }
    }
```

Override GetDebugInformation

```
//gets the debug information that is to be visually represented by the agents manager
public override DebugInformation[] GetDebugInformation()
{
    return debugInfo;
}
```