Building

Create a .Net assembly (Standard 2.0 preferred), include AntRunner.Interface.dll as a reference and create a class which inherts from AntRunner.Interface.Ant. Build your project and load the resulting DLL into AntRunner.exe.

See the ExampleAnt in the release zip file for an example of how to successfully create a basic ant.

Tips

- Implement structures to keep track of the map and your ant's current situation.
- Implement a path finding algorithm.
- Implement various strategies for your ant to use that can change depending on the GameState.
- Get your ant to successfully discover and navigate the map before worrying about battling other ants.
- Keep track of the last AntAction that way made so you know the direction you made an echo or moved for the next Tick().
- If you receive a GameEvent.CollisionDamage flag, do not update your ant's current location.
- You can do a lot of work in 250ms but O(n^2) algorithms will snowball quickly on a larger map.

Debugging

netstandard2.0 project is recommended.

Executing the AntRunner.exe with the argument debug causes the game manager to wait for each ant to finish before continuing so you have time to debug. You can also press F5 while AntRunner.exe is running to toggle debug mode on and off.

All AntRunner.exe arguments which are paths to dll files will attempt to load as ants at start up.

Setting up one click debugging for a .NET Standard project

1. Right click on your solution file and choose Add -> Exising Project....
2. Navigate to the AntRunner.exe file and open it.
3. Right click the AntRunner in solution explorer and choose Properties.
4. Under Arguments enter debug "C:\path\to\your\AwesomeAnt.dll". Save and close properties.
5. Right click the AntRunner in solution explorer and choose Set as StartUp Project.
6. Click Start Debugging and you will see your ant is loaded with the debugger attached. There will also be a red Debug in the upper right corner.

Setting up one click debugging for a .NET Framework project

1. Right click your project file and go to Properties and open the Debug tab.
2. Select Start external program.
3. Navigate to the AntRunner.exe file and open it.
4. Under Command line arguments enter debug "C:\path\to\your\AwesomeAnt.dll".
5. Under Workind directory enter the path to the AntRunner.exe.
6. Save and close properties.
7. Click Start Debugging and you will see your ant is loaded with the debugger attached. There will also be a red Debug in the upper right corner.

AntRunner.Interface

Ant

Base class for every ant, inherit from this class to create an ant.

```
public abstract class AntRunner.Interface.Ant
```

Example

```
public class AwesomeAnt : Ant
{
  //Your code here
}
```

Fields

Type	Name	Summary
AntAction	Action	Action which will be performed for the end of the current tick cycle. Will be set to Wait after being read.

Properties

Type	Name	Summary	Example
Stream	Flag	Overridable Stream to return binary data for use by BitmapFrame.Create() of the ant's Flag.	public override Stream Flag => typeof(Ant).Assembly.GetManifestResourceStream("AwesomeAnt.Flag.png");
String	FlagResource	Overridable string to return the name of an embedded resource for use the ant's Flag	public override string FlagResource => "AwesomeAnt.Flag.png";
String	Name	Readonly property for the name of the ant.	public override string Name => "Awesome Ant";

Methods

Type	Name	Summary
void	Initialize(Int32 mapWidth, Int32 mapHeight, ItemColor antColor, Int32 startX, Int32 startY)	Initialize method called once before the start of each game.
void	Tick(GameState state)	Method called to begin processing for each turn.

Initialize Example

```
public override void Initialize(int mapWidth, int mapHeight, ItemColor antColor, int startX, int startY)
{
   _mapWidth = mapWidth;
   _mapHeight = mapHeight;
   _myColor = antColor;
   _currentX = startX;
   _currentY = startY;
}
```

Tick Example

```
public override void Tick(GameState state)
{
   //Do Stuff
   Action = AntAction.MoveRight;
}
```

AntAction

Enum of available actions an ant can make per tick.

```
public enum AntRunner.Interface.AntAction
```

```
: Enum, IComparable, IFormattable, IConvertible
```

Enum

Name	Summary
Wait	Ant does nothing.
MoveRight	Ant attempts to move one space to the right.
MoveDown	Ant attempts to move one space below.
MoveLeft	Ant attempts to move one space to the left.
MoveUp	Ant attempts to move one space above.
EchoRight	Ant performs an echo to the right. The response will come in the GameState of the next Tick() call. See AntRunner.Interface.EchoResponse
EchoDown	Ant performs an echo below. The response will come in the GameState of the next Tick() call. See AntRunner.Interface.EchoResponse
EchoLeft	Ant performs an echo to the left. The response will come in the GameState of the next Tick() call. See AntRunner.Interface.EchoResponse
EchoUp	Ant performs an echo above. The response will come in the GameState of the next Tick() call. See AntRunner.Interface.EchoResponse
ShieldOn	Ant turns on its shield if it has any. For every 4 ticks the shield is on, it will lose 1 point.
ShieldOff	Ant turns off its shield.
DropBomb	Ant drops a bomb at the current position if it has any.
ShootRight	Ant shoot its laser in a straight line to the right.
ShootDown	Ant shoot its laser in a straight line below.
ShootLeft	Ant shoot its laser in a straight line to the left.
ShootUp	Ant shoot its laser in a straight line above.

DamageValues

Static class of constant values used when assigning damage to an ant.

```
public static class AntRunner.Interface.DamageValues
```

Static Fields

Name	Value	Summary
Bomb	30	Damage applied when an ant steps on a bomb.
Collision	5	Damage applied when an ant runs into an object.
Impact	10	Damage applied when an ant is run into or rammed by another ant.
Shot	20	Damage applied when an ant is shot with a laser.

EchoResponse

Response item when an Echo action is made.

```
public class AntRunner.Interface.EchoResponse
```

Properties

Type	Name	Summary
Int32	Distance	How many squares away is the item.
Item	Item	What item is there.

GameEvent

Enum flags of possible events that can occur as a result of all ants' GameAction. Multiple events may occur at once.

```
public enum AntRunner.Interface.GameEvent
    : Enum, IComparable, IFormattable, IConvertible
```

Enum

Name	Summary
Nothing	Nothing has occurred.
CollisionDamage	Ant ran into an object when it attempted to move and incurred damage. The move was unsuccessful and the ant remains at its current location. See AntRunner.Interface.DamageValues.Collision
ImpactDamageRight	Another ant ran into or rammed the ant from the right and incurred damage. See AntRunner.Interface.DamageValues.Impact
ImpactDamageDown	Another ant ran into or rammed the ant from below and incurred damage. See AntRunner.Interface.DamageValues.Impact
ImpactDamageLeft	Another ant ran into or rammed the ant from the left and incurred damage. See AntRunner.Interface.DamageValues.Impact
ImpactDamageUp	Another ant ran into or rammed the ant from above and incurred damage. See AntRunner.Interface.DamageValues.Impact
ShotDamageRight	Ant was shot by a laser and incurred damage from the right. See AntRunner.Interface.DamageValues.Shot
ShotDamageDown	Ant was shot by a laser and incurred damage from below. See AntRunner.Interface.DamageValues.Shot
ShotDamageLeft	Ant was shot by a laser and incurred damage from the left. See AntRunner.Interface.DamageValues.Shot
ShotDamageUp	Ant was shot by a laser and incurred damage from above. See AntRunner.Interface.DamageValues.Shot
BombDamage	Ant walked over a bomb and incurred damage. See AntRunner.Interface.DamageValues.Bomb
PickUpBomb	Ant picked up a Bomb Power-up. See AntRunner.Interface.ItemBonusValues.Bomb
PickUpShield	Ant picked up a Shield Power-up. See AntRunner.Interface.ItemBonusValues.Shield
PickUpHealth	Ant picked up a Health Power-up. See AntRunner.Interface.ItemBonusValues.Health
PickUpFlag	Ant picked up the Flag. Run to the correct color home!
Dead	Ant has died and will no longer be getting Tick() calls.
GameOver	The game is over, either an ant successfully retrieved the flag or all ants have died.

GameState

```
public struct AntRunner.Interface.GameState
```

Properties

Type	Name	Summary
ItemColor	AntWithFlag	If an ant has the flag, this is the color of that ant. If no ants have the flag then this value is ItemColor.None.
GameEvent	Event	Flag enum of which events occurred due to the previous Tick() GameAction.
Int32	FlagX	If an ant has the flag, this is their current X position on the map. If no ants have the flag then this value is -1.
Int32	FlagY	If an ant has the flag, this is their current Y position on the map. If no ants have the flag then this value is -1.
Boolean	HasFlag	Boolean value if an ant has the flag.
EchoResponse	Response	Echo response of the previous Tick GameAction. If previous Tick was not an echo action then this is null. See AntRunner.Interface.AntAction.EchoRight AntRunner.Interface.AntAction.EchoLeft AntRunner.Interface.AntAction.EchoUp AntRunner.Interface.AntAction.EchoDown
Int64	TickNumber	Long value of the tick turn, starts with 0 and each Tick() call is increased by 1. See AntRunner.Interface.Ant.Tick(AntRunner.Interface.GameState)

Item

Enum of possible items on a map position.

```
public enum AntRunner.Interface.Item
    : Enum, IComparable, IFormattable, IConvertible
```

Enum

Name	Summary
Empty	Nothing at all, aka empty.
SteelWall	Unbreakable steel wall.
BrickWall	Brick wall which may be destroyed with a laser shot.
Bomb	Bomb that will damage the ant if stepped on. Can be shot with the laser to destroy See AntRunner.Interface.DamageValues.Bomb
PowerUpBomb	Bomb Power-up, adds bombs to the ant's inventory. See AntRunner.Interface.ItemBonusValues.Bomb
PowerUpHealth	Health Power-up, adds more health level to the ant, maximum 100. See AntRunner.Interface.ItemBonusValues.Health
PowerUpShield	Shield Power-up, adds more shield level to the ant, maximum 100. See AntRunner.Interface.ItemBonusValues.Shield
RedAnt	The red ant.
BlueAnt	The blue ant.
GreenAnt	The green ant.
OrangeAnt	The orange ant.
PinkAnt	The pink ant.
YellowAnt	The yellow ant.
GrayAnt	The gray ant.
WhiteAnt	The white ant.
RedHome	The home for the red ant.
BlueHome	The home for the blue ant.
GreenHome	The home for the green ant.
OrangeHome	The home for the orange ant.
PinkHome	The home for the pink ant.
YellowHome	The home for the yellow ant.
GrayHome	The home for the gray ant.
WhiteHome	The home for the white ant.
Flag	The flag, pick this up and bring it to the correct color home.

ItemBonusValues

Static class of constant values used when assigning power-ups to an ant.

```
public static class AntRunner.Interface.ItemBonusValues
```

Static Fields

Name	Value	Summary
Bomb	4	Amount of bombs added to an ant's inventory when a bomb power-up is acquired. See AntRunner.Interface.Item.PowerUpBomb
Health	25	Amount of health that is restored when a health power-up is acquired. See AntRunner.Interface.Item.PowerUpHealth
Shield	25	Amount of shield that is restored when a shield power-up is acquired. See AntRunner.Interface.Item.PowerUpShield

ItemColor

Enum of the available ant and home colors.

```
public enum AntRunner.Interface.ItemColor
    : Enum, IComparable, IFormattable, IConvertible
```

Enum

Name	Summary
None	No color at all.
Red	Color for the RedAnt and RedHome
Blue	Color for the BlueAnt and BlueHome
Green	Color for the GreenAnt and GreenHome

Orange	Color for the OrangeAnt and OrangeHome
Pink	Color for the PinkAnt and PinkHome
Yellow	Color for the YellowAnt and YellowHome
Gray	Color for the GrayAnt and GrayHome
White	Color for the WhiteAnt and WhiteHome

Maps

Maps can be created by making a simple bitmap (bmp) image file and placing it in the Maps folder. Each pixel is a point on the map. All colors which are not defined are ignored.

SteelWall

Black $\mathrm{rgb}(0,0,0)$

Do not create empty spaces completely surrounded by SteelWall. It is possible for an ant/home/flag to randomly be placed in this space. The edge of the map will always be a SteelWall so you do not need to draw this in.

BrickWall

Red $\mathrm{rgb}(255,0,0)$

There should always be some included because the only way to obtain power-ups is by shooting BrickWalls.

AntHome

Blue $\mathrm{rgb}(0,0,255)$

If there aren't enough home locations on the map for the ants loaded then homes will be randomly placed. Colors are randomly assigned to each home location. There can be more possible home locations than 8 but there will only be homes allocated for as many ants loaded.

Flag

Green $\mathrm{rgb}(0,255,0)$

If there is no flag location set then the flag will randomly be placed. There can be multiple possible flag locations and a location will be chosen at random.