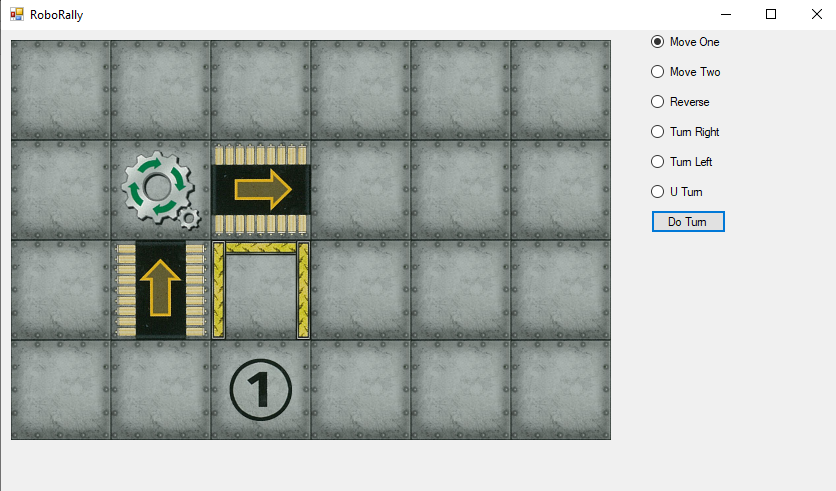
# Assignment 4: RoboRally

This assignment focusses on object oriented programming, image manipulation, and a combination of inheritance and composition to create a complex set of classes to build a world where ‘robots’ move around a factory floor. This is an actual boardgame that has been implemented multiple times as a computer game or app; we’re building out a subset of the behaviour.

The RoboRally Board Looks as follows:  
  


The circled ‘1’ represents the robot. The robot can be given a command each turn, and will follow that command. After the robot follows its orders (if possible – robots cannot walk through walls), various board spaces will then perform their own actions. In the above example there are two conveyor belts that move robots and a gear that will spin a robot that is on top of it.

Move the robot around the board to see the conveyor belts and gears in action, and re-assure yourself that robots cannot walk through walls.

There are several important classes that you’ll want to understand:

* RobotControls: Panel
  + This extends the build-in Panel object, adds the ‘Do Turn’ button in the constructor, and allows a way to add other robot actions to build the list of radio buttons
* Board
  + A Board.Width x Board.Height playing surface with spaces that are Board.SpaceSize pixels square.
  + Each spot on a board is a BoardSpace object
* BoardSpace
  + Has a set of features and knows where it is on the Board.
  + Can draw all of its features based on their image and their rotation.
* Abstract Feature
  + An abstract class that has an image, a set of actions, a location, a direction, etc.
  + Some key virtual functions are whether a feature blocks movement or light (there are lasers in the actual game, unimplemented here), and whether the feature itself is moveable
* Robot : Feature
  + A robot is a moveable feature (that has its own action that is determined by the RobotControls panel)
* StationaryFeature: Feature
  + A base class for all non-moveable features
* Floor: StationaryFeature
  + Boring feature that nothing but an image
* Gear: StationaryFeature
  + A space could have a floor and a gear on it – the gear turns moveable objects on the same space
* Conveyor: StationaryFeature
  + A space that moves anything moveable on it one space
* Wall: StationeryFeature
  + A wall blocks movement and light
  + You can add multiple walls on a space
* Action
  + An abstract representation of an action that can occur on the board.
  + Features can have a set of actions (and these require priorities so we know which go first
  + Robots have an action
* Turn: Action
  + Turn something left, right, or u-turn
  + Gears turn things on them left or right
  + Robots can get a left, right, or u-turn action that applies only to them.
* Move: Action
  + Move something in a certain direction a certain distance
  + Robot commands move in the direction the robot is facing either one space, two spaces, or -1 spaces (reverse)
  + Feature commands move anything moveable on them in a certain direction.
* Enum Rotation
  + There are three rotations – clockwise (right), widdershins (counter-clockwise, or left), and uturn (180 degrees)
* Static Direction
  + This is a static class with four points representing the directions Up, Down, Left, Right
  + It also has a ‘spin’ function which takes a rotation and a direction and gives back the new direction that the object should be facing.

## How the game works

Roborally is played in a sequence of ‘rounds’, each consisting of 5 ‘turns’. Here we don’t have ‘rounds’, just a series of ‘turns’. A ‘turn’ happens when the user picks a command and then clicks the ‘do Turn’ button.

The robot will attempt to execute its action.  
  
Then all the actions on the board setup check, in priority order, if they can be performed, and if so, affect the robot (or really, anything moveable)

(We check all actions rather than just for the space the robot is on because in a full game of roborally there might be several robots, there could be moving boxes, and some actions are non-local (e.g. a laser which shoots over several squares but would be blocked by a wall or a robot).

## Image Manipulation

There is a file called 2005TileSetA.png that has all the images that are required for the game.

Une image contenant carré

Description générée automatiquement

There is a static class called RoboRallyImageSet that is built to get individual images out of the file. From the image above, you’ll notice that the background is a hot pink – that is used to set the transparency colour, so anything that is pink in the file image will be transparent when you draw it on the board. There is an ‘extract’ function that takes the location of the sub-image (conveyor belt, gear, wall, etc.) in the file, pulls out the image, then makes the pink bit transparent. In the case of the wall, it then rotates it, so that the default ‘Up’ wall is a wall at the top of a square.

Each Feature has an image and a direction. So for Features where the direction is ‘Up’, the images are drawn on the board. If the direction is Left, Right, or Down, they are rotated, and then drawn. You can see that for the right-rotated conveyor belt and the BoardSpace that has 3 walls.

In the underlying file, each image is 300 x 300 pixels – and we draw the board to an image that as 300 x 300 board spaces. Whenever we draw the board, we create a new bitmap image, draw the board on that image, and \*then\*, only once the image is fully built, do we set the new image as the image in the PictureBox – the PictureBox renders the image according to the scale we set on the picturebox.

## Core Task: Add Outside Walls (1 pt)

Right now, you can walk the robot off the board, causing an exception. We could add exception handling, but there is a more straightforward way to prevent the error. Add walls all the way around the board so that a robot cannot walk off the board. (Hint A)

## Core Task: Add A Turn Counter (1 pt)

You’ll notice the image in tileset 2,3 is a ‘Pusher’. It will push anything in front of it in the direction it is facing. But in order for that, we need a publically accessible Turn property on the Board class that counts what turn it is. It should be a public getter but a private setter, and you’ll want to add a ‘next Turn’ function on Board that adds one to the turn (Hint: that would itself get called from the doTurn\_Click handler. The first turn of the game is turn 1. After turn 5 it is not turn 6, but rather Turn 1 again. Also add a way to display the current turn somewhere.

## Core Task: Create a Pit Class (2 pts)

In Roborally there are ‘pits’. Conceptually, a robot can fall in them, but cannot get out. In the roborally world, a pit is a space that you can move into, but cannot leave. That is, its ‘blocksMove’ function should block movement in every direction, but only when leaving.

There are several steps to setting up a Pit in the game:

* Add a property to RoboRallyImageSet
  + Static public Image Pit {get; set;} (Hint B1)
  + Extract the image from tileset at co-ordinates 2,4 (Hint B2)
* Create a Pit Class
  + Model it on the Wall class (Hint B3)
  + Pits don’t block light (Hint B4) – e.g. a laser would fire right over it
  + Pits don’t block movement entering the pit, but do block all movement exiting (Hint B5)
* Add a Pit to the Map
  + Create a Pit on the Board in the Board constructor (Hint B6)
* Test and make sure that your robot can walk (or get conveyored) \*into\* a pit, but can’t get out.

## Core Task: Create A Left Turn and a Right Turn Conveyor (3 pts)

There is already a conveyor class, but it assumes that every conveyor belt moves whatever is on it exactly one space in the direction of the arrow, with no rotation. In position 0,1 of the tileset, there is a conveyor belt that takes whatever is on it, turns it left, and then moves it forward a space.

You already have a conveyor class, and we aren’t going to change that, we just have three types of conveyors – straight, right, and left. The straight conveyor only has a single action (move 1), but left and right will need to do a \*turn\* and \*then\* a move.

Again, there are several bits to get this to work:

* You’ll need ConveyorLeft and ConveyorRight properties (Hint C1)
* Create a Conveyor left image (Hint C2)
* Create a Conveyor right image (Hint C3)
* Enhance the Conveyor class to take an argument of what type of conveyor it is
  + Likely a rotation? (Hint C4)
  + Make sure you add both actions (Turn and Move) and get the priorities right
* Add both new types of conveyors to the map (Hint B6 – same to add conveyors as pits)
* Test and make sure that the conveyor turns and moves anything that is on it.

## Core Task: Build a Winner! Square (2 pt)

Build a Winner class with the image in 3,1 (crossed wrench and hammer). It is basically a floor space with a different image and its own \*unique\* action. You’ll want to create a new action that determines if there is a robot (Hint: check for anything moveable, and then check to see if the Feature f is a robot ( if f is Robot)). If there is, it should pop up a MessageBox that says Winner!!!

## Core Task: Build a More Interesting Map (1 pt)

Now that you have left and right turn conveyors, pits, and a winning square, create your own custom map by putting together multiple copies of the component squares in an interesting way. Submit a screenshot of your custom map.