Demonstration: Tetris

Tetris was originally developed by Alexey Pajitnov. The rights are currently held by Tetris Company. This demonstration and its code are purely for demonstrative purposes and no infringement is intended.

This demonstration gives step-by-step, recipe-style instructions on building a Tetris game using the hive system. ***This is not a small thing***: in pure Python, a typical implementation of a Tetris game is about 500 lines of code, and it may take you a day or two if you can program (and weeks or months if you are still learning). The hive version has 141 nodes and about 70 lines of Python code. A video version of this demonstration is planned for the future and may be available when you read this. Also, a non-visual (Python hive) implementation of Tetris is explained in the hive system manual.

This demonstration does **not** explain how the hive system works. It assumes that you have a working installation of the hive system (and Spyder, Qt, Blender/Panda3D, …) , and a very basic knowledge of the HiveGUI. To get to that level, see the installation instructions and chapter 1 of the tutorial.

The demonstration is divided into 12 steps. Unfortunately, the first steps are the ***hardest***: the later steps are considerably easier. In the hive system, it is apparently easier to modify an existing setup than to create a new one from scratch. If you find the first steps too difficult, you can skip them: for example, you can take the “tetris5/” folder as starting point and start with step 6.

Step 0: Getting started

Download the default empty hive project at https://launchpad.net/hivesystem/+download and unzip it.
Rename the folder from “defaultproject/” to “tetris/”. Go into the folder.
Remove “defaultproject.py”.

If you want to run Tetris under Blender
Copy or rename “defaultproject.hivemap” to “tetrishive.hivemap”
Rename “defaultproject-blender.py” to “tetrishive.py”. Open “tetrishive.py”. In line 9, change the text "defaultproject.hivemap" to "tetrishive.hivemap" and save it.
Rename “defaultproject.blend” to “tetrishive.blend”. Open “tetrishive.blend” with Blender. In the Text Editor, press the X button. Press Alt+O and open “tetrishive.py”. In the Properties Editor, select the Scene tab and navigate to Custom Properties. Edit the “__main__” property and change it from “defaultproject-blender.py” to “tetrishive.py”. In the main Blender menu, go to File => External Data => Make All Paths Relative. Save the .blend.

If you want to run Tetris under Panda3D
Copy or rename “defaultproject.hivemap” to “tetrishive-panda.hivemap”
Rename “defaultproject-panda.py” to “tetrishive-panda.py”. Open “tetrishive-panda.py”. In line 9, change the text "defaultproject.hivemap" to "tetrishive-panda.hivemap" and save it.

Remove “defaultproject.hivemap” if you haven't done so already.
To run Tetris under Blender, open “tetrishive.blend” and press P. Under Windows, be sure to have the system console enabled.
To run Tetris under Panda3D, run “tetrishive-panda.py”.
In either case, press Esc to end the hive.
This is true for all steps in this demonstration.

You can find the completed project of step 0 in the “tetris0/” folder.

Step 1: Setting up a canvas, and displaying the score

First, we will define a drone to draw the *canvas*, the 2D area upon we will draw all shapes. For more information on this, see chapter 2 of the tutorial.
- *If you want to run Tetris under Blender*
Open “tetrishive.hivemap” with HiveGUI. Under Drones, select dragonfly => blenderhive => blendercanvas and create a new node. Rename the name (Worker ID) of the new node from “blendercanvas_1” to “canvas”. Save the hivemap and quit HiveGUI.
- *If you want to run Tetris under Panda3D*
Open “tetrishive-panda.hivemap” with HiveGUI. Under Drones, select dragonfly => pandahive => pandacanvas and create a new node. Rename the name (Worker ID) of new the node from “pandacanvas_1” to “canvas”. Save the hivemap and quit HiveGUI.
- From now on, all instructions, in all of the steps, are the same for Blender and Panda3D. Therefore, from now on, “tetrishive.hivemap” or “tetrishive-panda.hivemap” will simply be called “the tetrishive”.

Second, we will create a simple worker-like hivemap that can draw the score. It will have one parameter, holding the identifier of the score area, and two antennas: an pull-integer antenna for the value of the score, and a push-trigger antenna to activate the drawing.
- Open HiveGUI. Create a dragonfly.canvas.draw3 worker. Set type to “str”. Rename the worker to “w_draw”.
- Create a bees.parameter and rename it to “scorearea_id”. Set the internal name to “scorearea_id_” (note the final underscore). Set the typename to “str”.
- Create a wasp that will insert the value of “scorearea_id” into the w_draw worker's “identifier” parameter. For the wasp, set “injected” to “scorearea_id_”, the target name to “w_draw”, and the target parameter to “identifier”.
- Create a worker to convert “int” to “str”: create a dragonfly.convert.pull.cast worker, set type1 to “int” and type2 to “str”. Rename the worker to “get_score”.
- Connect “get_score” to “w_draw”. In the pop-up menu, select “manual”. Rename the transistor to “set_score”.
- Create a bees.io.pull_antenna, name it “score”, and connect it to get_score
- Create a bees.io.push_antenna, name it “draw”, and connect it to set_score.trig
Save the hivemap as “hivemaps/tetris_draw_score.hivemap”. Quit HiveGUI

Third, we will define a *main hivemap* (tetrismain). The main hivemap will contain a score that is initially zero. The score will be drawn using the simple “tetris_draw_score” hivemap that we have just defined.
Open HiveGUI. Save the new, empty hivemap as “hivemaps/tetrismain.hivemap”. Go to File=>Refresh. Now, under workers, there should be an entry “hivemaps” => “tetris_draw_score”. If not, quit HiveGUI and open the main hivemap again.
- Create a new tetris_draw_score node. Name it “tetris_draw_score”. It should have two antennas: a (“push”, “trigger”) antenna called “draw” and a (“pull”, “int”) antenna called “score”. It should have a single parameter called “scorearea_id”. Again, this parameter should be defined by the parent hive. Therefore:
- Create a bees.parameter, name it “scorearea_id”. Set its internal name to “scorearea_id_” and its type to “str”. Create a new wasp, set “injected” to “scorearea_id_”, the target name to “tetris_draw_score”, and the target parameter to “scorearea_id”.

- Create an int variable named “score” to hold the score. Set its value to 0. Connect it to tetris_draw_score.
NOTE: In the score variable, be sure to go to the value field, click on the 0 and press Enter. A 0 should appear also on the canvas. The hive system discriminates between 0 and “undefined”, but both are currently rendered as “0” in the HiveGUI.
Finally, save the hivemap and quit HiveGUI.

Fourth, we will congure the *score area*, the screen area where the score will be drawn.
- Open SpyderGUI. On top of the screen, set the parent class to “bee.spyderhive.spyderframe”.
- Specify that the score area needs a canvas parameter. Simply create a parameter and name it “canvas”.
- Create a new attribute with spydertype = “CanvasSlot” and name (Bee ID) = “scorearea” (no quotes).
In the parameters:

- Set “canvasname” to “canvas” (no quotes)
- Set “slotname” to “score”
- Set “slottype” to “str”
- Define the box with x=170, y = 100, sizex = 80, sizey = 40.

Save the file as “spydermaps/area.spydermap”. Make sure that the message “Warning: attribute 'scorearea' (CanvasSlot) has no value, skipped!” does not appear: if it does, you made a mistake in editing it. After it works, quit SpyderGUI.

Fifth, we will embed the main hivemap and the score area in the tetrishive. Open the tetrishive again with HiveGUI. We have previously created a canvasdrone called “canvas”, it should still be there.
- In “spydermaps”, an entry “area” should appear. Create a new area and rename it “area”
- Supply the canvas drone as the canvas parameter to “area”. Create a wasp, set “injected” to “canvas”, the target name to “area” and the target parameter to “canvas”.
- In “hivemaps”, an entry “tetrismain” should appear. Create a new “tetrismain” node and set its “scorearea_id” to “score”. “score” is the name of the CanvasSlot that we defined in the spydermap.

Save the tetrishive and run your hive. You will see the score “0” printed on the screen.

You can find the completed project of step 1 in the “tetris1/” folder.

Step 2: Drawing a grid for the blocks

In this step, we will define a grid where the Tetris blocks will be drawn. The grid will be a binary grid (bgrid), a 2D grid of which the elements can be True (occupied) or False (empty). True will be rendered as white, False as black. Python programmers can look at the bgrid source code in “dragonfly/grid/bgrid.py”. This Python class is known by the hive system as (“object”, “bgrid”)
We will define a bounding box area, and a simple hivemap that will draw the grid inside this area.

First, we will define a simple worker-like hivemap to draw the grid. The hivemap will be called “tetris_draw” and will have a parameter “mainarea_id”, holding the identifier of the mainarea bounding box, and two push-trigger antennas: “start” to do the initial drawing, and “draw” to re-draw the grid when some of its elements have changed.
- Open HiveGUI
- Create a variable of the type (“object”, “bgrid”) and name it “drawgrid”
- The bgrid has a Spyder model, called Bgrid, for visual editing. We will define an empty Bgrid to fill the value of “drawgrid”. Create an attribute (bees.attribute) of spydertype Bgrid and name it

“emptygrid”. Explicitly set all elements of “drawgrid” (minx, maxx, miny, maxy) to 0, by clicking on the 0s and pressing Enter. Create a wasp to fill “drawgrid”: set “injected” to “emptygrid”, target name to “drawgrid”, target name to “value”, and enable the “sting” option.
- Create a dragonfly.canvas.draw3 worker, set the type to (“object”, “bgrid”), and rename it to “w_draw”. Connect “drawgrid” to “w_draw”. In the pop-up menu, choose “manual”. Rename the auto-created transistor to “do_draw”.
- Create an update3 worker (dragonfly.canvas.update3) and rename it to “update”.
- Define a bees.parameter called “mainarea_id”, set the internal name to “mainarea_id_” and the type name to “str”.
- Define two wasps that set the value of “mainarea_id” as the identifiers of “drawgrid” and “update”: set “injected” to “mainarea_id_”, the target parameter to “identified”, and the target names as “drawgrid” and “update”.
- Create a bees.io.push_antenna “start” and connect it to do_draw.trig.
- Create a bees.io.push_antenna “draw” and connect it to update.
- Save the hivemap as “hivemaps/tetris_draw.hivemap”
- To make sure that the Bgrid class is imported by the main script, edit “spydermodels/__init__.py” and add the following code:

```
import spyder
import dragonfly.grid
```

Second, we will add the drawing of the grid to the main hivemap. This will also add a new parameter “mainarea_id” to the main hivemap, in addition to “scorearea_id”.
- Open “hivemaps/tetrismain.hivemap” with HiveGUI.
- In “hivemaps”, an entry “tetris_draw” should appear. Create one, and rename it to “tetrisdraw”.
- Draw the grid upon startup. Create a startsensor (dragonfly.sys.startsensor), name it “start”. Connect it first to tetrisdraw.start, and then to tetrisdraw.draw.
- Select the scorearea_id parameter, and create a copy. Rename the copy to mainarea_id. Rename its internal name to “mainarea_id_”
- Create a new wasp, setting “injected” to “mainarea_id_”, target name as “tetrisdraw”, and target parameter as “mainarea_id”.
- Save the hivemap.

Third, we will define a bounding box area for the main grid
- Open “spydermaps/area.spydermap” with SpyderGUI.
- Create a new attribute “mainarea” with spydertype = “CanvasSlot” (or you can copy “scorearea” and edit the copy). In the parameters:
  - Set “canvasname” to “canvas” (no quotes)
  - Set “slotname” to “main”
  - Set “slottype” to (“object”, “bgrid”)
  - Define the box with x=100, y = 150, sizex = 225, sizey = 375
  - Define the color as r = 0.5, g= 0.5, b = 0.5, a=0

Save the file. Again, make sure that there is no message “Warning: attribute 'mainarea' (CanvasSlot) has no value, skipped!” before quitting SpyderGUI.

Finally, we will pass on the bounding box to the main hivemap from within in the tetris hive
- Open the tetrishive with hivegui. The tetrismain worker should now have a new parameter called “mainarea_id”. Fill in “main”, the slotname that we just defined in the “area” spydermap.

Save the tetrishive and run your hive. You will now see a black area where the empty grid is drawn.

You can find the completed project of step 2 in the “tetris2/” folder.

Step 3: Defining the grid

The contents of a Tetris box consist of the current block, plus the (remainder of) previous blocks that do not yet form completed horizontal lines. Therefore, throughout this demonstration, we will define three different grids:
- The **drawgrid**, defined in step 2, contains what the user sees: both the current block and the previous blocks.
- The **maingrid**, which will be defined here, contains the previous blocks.
- The **blockgrid**, which will be defined in step 4, will contain the current block

First, we will modify the tetris_draw hivemap to render the contents of the maingrid into the drawgrid. This will be done using bgridcontrol (dragonfly.grid.bgridcontrol) workers.
- Create a bgridcontrol and name it “drawgridcontrol”. Connect the drawgrid to drawgridcontrol.grid.
- Create a second bgridcontrol called “maingridcontrol”.
- Create a pull antenna (bees.io.pull_antenna), name it “maingrid”, and connect it to maingridcontrol.grid.
- Connect maingridcontrol.copy to drawgridcontrol.set. In the pop-up menu, choose “manual”. Rename the auto-created transistor to “copy_maingrid”
- Remove the connection between “draw” and “update”. Create a trigger-connector (dragonfly.std.pushconnector, type=”trigger”) and name it “trigger”. Connect “draw” to trigger.inp. Connect trigger.outp to copy_maingrid.trig, and then connect trigger.outp to “update”.
Save the hivemap.

Second, we will specify the maingrid in the “tetrismain” hivemap. Open it with HiveGUI. The “tetrisdraw” node should now have an additional (“pull”, (“object”, “bgrid”)) antenna called “maingrid”.
- Define a new (“object”, “bgrid”) variable called “maingrid”. Connect it to tetrismain.maingrid.
NOTE: the HiveGUI may not recognize their types to be the same, and refuse to make a connection. This is a bug. If this happens, keep the CTRL key pressed while you connect the nodes. This forces the connection.
- Fill the maingrid with an initial value. Create a new bees.attribute of spydertype = “Bgrid”, named “maingrid_init”. Set minx to 0, maxx to 9, miny to 0 and maxy to 19. Again, make sure to enter the 0s explicitly. Create a wasp to insert the value into “maingrid”. Set “injected” to “maingrid_init”, the target name to “maingrid”, and the target parameter to “value”. Don't forget to enable the sting option.

Save the hivemap and run your hive. You will now see a 10x20 empty grid filled with black squares.
You can find the completed project of step 3 in the “tetris3/” folder.

Step 4: Adding a tetris block to the grid

Now we will define the blockgrid.
First, open the tetris_draw hivemap. Select the “copy_maingrid” transistor, copy it, and rename the copy to “t_blockgrid”. Connect t_blockgrid to drawgridcontrol.merge. Select the “maingrid” antenna node, copy it and rename the copy to “blockgrid”. Connect blockgrid to t_blockgrid. Connect trigger.outp to t_blockgrid.trig. Now, make sure that the connections of trigger.outp are in the correct

order. The first (most upwards-going) connection must be to copy_maingrid. The second (middle) connection must go to t_blockgrid. The final (most downwards-going) connection must be to “update”. You can disconnect and re-connect until the order is right. Or you can hover the mouse over the red dot of trigger.outp, press TAB repeatedly to highlight a connection, and then press 1,2 or 3 to reorder them. Finally, save the hivemap.

Second, open the tetrismain hivemap. The “tetrisdraw” node should now show an additional (“pull”, (“object”, “bgrid”)) antenna called “blockgrid”.
- Select the “maingrid” variable and copy it. Rename the copy to “blockgrid”. Connect the blockgrid to tetrisdraw.blockgrid. Again, keep the Ctrl key down if the connection is refused.
- Create a new attribute of spydertype “Bgrid” and rename it to “blockL”. Edit its value, but leave minx,miny,maxx and maxy untouched. In “Values”, set “0” to x=0,y=0 (again, explicitly enter the 0 each time). Set “1” to x=1, y=0. Set “2” to x=2, y=0. Finally, set “3” to x=0,y=1.

Save the hivemap and run the hive. You should see an L-shaped Tetris block in the lower-left corner. If the grid is empty and there is no error message, there are two likely causes:
1. The editing of blockL went wrong. Open the tetrismain hivemap and see if the values were entered and saved correctly, try again, and repeat. If you can't get it right, try to delete blockL and create it again.
2. The connection order of trigger.outp was specified incorrectly. Open tetris_draw and check that the order is correct.

You can find the completed project of step 4 in the “tetris4/” folder.

Step 5: Random selection and orientation of the block

First, we will create a worker-like hivemap that can select a random block and put it into a random orientation. The random block will be selected from a list. A random orientation will be achieved by performing 90 degrees rotations up to four times in a row.
- Start HiveGUI
- Create a parameter (bees.parameter) named “blocks”. As the internal name, enter “blocks_”. As the type, enter (“object”, “array”, “bgrid”).
- Create a variable of type “object” named “w_blocks”. Define a wasp that inserts the contents of “blocks” into “w_blocks”: setting “injected” to “blocks_”, target name to “w_blocks” and target parameter to “value”.
- Create a dragonfly.random.choice worker and name it “sel”. Connect w_blocks to sel.
- Create a transistor with type=(“object”, “bgrid”) named “do_select”. Create a variable with type=(“object”, “bgrid”) named “chosen”. Connect “sel” to “do_select”. Connect “do_select” to “chosen”.
- Create a bgridcontrol (dragonfly.grid.bgridcontrol) and name it “chosencontrol”. Connect “chosen” to chosencontrol.grid.
- Create an variable with type=int and rename it to “four”. Set its value to 4.
- Copy “four”, rename the copy to “zero”, and set its value to 0.
- Create a weaver (dragonfly.std.weaver) and set its type to (“int”,”int”). Name it “uptofour”. Connect “zero“ to inp1 and “four” to “inp2”.
- Create a randint worker (dragonfly.random.randint) named “randint”. Connect “uptofour” to randint.
- Connect “randint” to chosencontrol.rotate. In the pop-up menu, select “manual”. Rename the auto-created transistor to “rotate”.
- Create a trigger connector (dragonfly.std.pushconnector, type=trigger) named “trigger”. Connect

trigger.outp first to do_select.trig, then to rotate.trig.
- Create a bees.io.push_antenna named “select”, and connect it to trigger.inp.
- Select the “do_select” worker, copy it, and rename the copy to do_select2. Connect “chosen” to do_select2. Connect trigger.outp to do_select2.trig.
- Create a bees.io.push_output named “selected”. Connect do_select2.outp to selected.
Save the hivemap as hivemaps/tetris_select_block.hivemap

Second, we will add the block selector to the tetrismain hivemap.
- Open the tetrismain hivemap with HiveGUI. Under “hivemaps”, an entry “tetris_select_block” should be visible. Create a “tetris_select_block” node and rename it “select_block”. In its parameters, there should be an entry “blocks”.
- Create a “blocks” parameter (bees.parameter), with internal name “blocks_” and typename (“object”, “array”, “bgrid”).
- Create a wasp that passes on the “blocks” parameter to “select_block”. Set “injected” to “blocks_”, target name to “select_block” and target parameter to “blocks”.
- Connect start.outp to select_block.select. Adjust the connection order of start.outp: the first (most upwards-going) connection must go to select_block.select, the middle one to tetrisdraw.start, the final (most downwards-going) connection to tetrisdraw.draw.
- Rename the attribute “blockL” to “emptygrid”. Select “emptygrid”, go to Parameters => Metaparams, and press “Create Instance”, this will reset its value. In the wasp that targets blockgrid, change “injected” from “blockL” to “emptygrid”.
- Create a bgridcontrol (dragonfly.grid.bgridcontrol) named “blockgridcontrol”. Connect “blockgrid” to blockgridcontrol.grid. Connect “select_block” to blockgridcontrol.set. Again, hold the Ctrl key if the HiveGUI refuses the connection.
- Create a keyboardsensor (dragonfly.io.keyboardsensor_trigger). Set its keycode to SPACE. Connect the keyboardsensor to “select_block”, then to tetris_draw.draw.
Save the hivemap.

Third, we will define the blocks inside the tetrishive.
- Open the tetrishive hivemap with HiveGUI.
- Select “tetrismain”. In “Parameters”, a new entry “blocks <type = (“object”,”array”,”bgrid”)> should appear.
- Create a new attribute of spydertype “Bgrid” and rename it to “blockL”. Edit its value, but leave minx,miny,maxx and maxy untouched. In “values”, set “0” to x=0,y=0 (again, explicitly enter the 0 each time). Set “1” to x=1, y=0. Set “2” to x=2, y=0. Finally, set “3” to x=0,y=1.
- Create a wasp to ***add*** blockL to the “blocks” parameter of “tetrismain”. To do so, enable the “accumulate” option of the wasp. As “injected”, specify “blockL”. As target name, specify “tetrismain”, and the target parameter is “blocks”. Don't forget to enable the “sting” option.
- Copy blockL and rename it to blockI. Edit its value: in “values”, change “3” from x=0,y=1 to x=3,y=0.
- Copy the last wasp and edit the copy, changing “injected” from “blockL” to “blockI”.
Save the hivemap. Re-open it and inspect blockL and blockI if their x,y values are still correct. Then, run the hive. You will see an L-block or an I-block in a random orientation in the left bottom corner (although part of it may be off the grid and invisible). Pressing SPACE will select a new block in a new orientation.

You can find the completed project of step 5 in the “tetris5/” folder.

Step 6: Defining all Tetris blocks

We will now define all Tetris blocks, but not as the cumbersome (x,y) pairs of Spyder.Bgrid. Instead, we will define a more convenient Spyder model, TetrisBlock, that allows more intuitive visual editing.

Open up a text editor and start a new file that will contain our Spyder model (a Python editor like IDLE will also do; Spyder isn't exactly Python, but close enough). This code defines the data layout:

```
Type TetrisBlock {
  BoolArray col0[4] = (False,False,False,False)
  BoolArray col1[4] = (False,False,False,False)
  BoolArray col2[4] = (False,False,False,False)
  BoolArray col3[4] = (False,False,False,False)
}
```

To make the editing nicer, specify that the editing form must use a column layout:

```
Type TetrisBlock {
  BoolArray col0[4] = (False,False,False,False)
  BoolArray col1[4] = (False,False,False,False)
  BoolArray col2[4] = (False,False,False,False)
  BoolArray col3[4] = (False,False,False,False)
  form {
    self.subtype = "column"
  }
}
```

Finally, we will add a converter of TetrisBlock to Bgrid, consisting of Python code:

```
Define Bgrid(TetrisBlock b) {
  values = []
  for x,col in enumerate([b.col0,b.col1,b.col2,b.col3]):
    for y,ele in enumerate(col):
      if ele: values.append((x,y))

  if len(values) == 0:
    grid = Bgrid (
      minx = 0,
      maxx = 0,
      miny = 0,
      maxy = 0,
    )
  else:
    grid = Bgrid (
      values = values
    )
  return grid
}
```

Save the file as spydermodels/tetrisblock.spy.

To import the Spyder model when we run the hive, add the following code to spydermodels/__init__.py:

```
from . import tetrisblock
```

Finally, restart HiveGUI and open the tetrishive hivemap. The existing attributes with spydertype=Bgrid, which are blockL and blockI, still work. But in addition, you can now create attributes with spydertype=TetrisBlock. Create such an attribute and name it blockO. Edit its value and define the four bottom-left elements to be filled. Then, create a wasp to add it to the blocks, just as before: injected = blockO, target_name = tetrismain, target_parameter = blocks, accumulate and sting are enabled. Finally, create all Tetris blocks in this manner: I, J, L, O, S, T and Z.
Save the hivemap and run the hive. Press SPACE to cycle through all blocks and orientations.

You can find the completed project of step 6 in the “tetris6/” folder.

Step 7: placing the current block

Now we will move the current block to the top middle of the maingrid. Since this involves some Python, we will create a custom workermap, using the WorkerGUI.

- Start the WorkerGUI.
- Create a pull antenna (segments.antenna.pull_antenna) of type (“object”, “bgrid”), and name it “maingrid”. Then, create a pull buffer (segments.buffer.pull_buffer) of type (“object”, “bgrid”), and name it “grid1”. In its parameters, set “triggerfunc” to “get_grid1”. Connect “maingrid” to “grid1”.
- Draw a rectangle around “maingrid” and “grid1”, selecting them both. Copy them, and in the copy, rename the copy of “maingrid” to “blockgrid” and the copy of “grid1” to “grid2”. In “grid2”, change “triggerfunc” to “get_grid2”.
- Create a node for custom class code (segments.custom_code.custom_class_code) and name it “get_grids”. In its parameters => code field, enter the following Python code:

```
def get_grids(self):
  self.get_grid1()
  self.get_grid2()
```

- Create a trigger antenna (segments.antenna.push_antenna_trigger) called “place_init”.
- Create a modifier (segments.modifier.modifier) named “m_place_init”. In its parameters => code field, enter the following Python code:

```
self.get_grids()
dx = int(self.grid1.maxx/2)-self.grid2.minx
self.grid2.maxx += dx
self.grid2.minx += dx
dy = self.grid1.maxy - self.grid2.maxy
self.grid2.maxy += dy
self.grid2.miny += dy
```

- Connect “place_init” to “m_place_init”.
- Save the workermap as “workermaps/tetris_control.workermap”. Now, in the menu, go to Generate

=> Generate code, or press Ctrl+G. Copy-paste the generated Python code into a text editor and save it as “workers/tetris_control.py”.

- Open the tetrismain hivemap with HiveGUI. On the left side, under “dragonfly”, an entry “workers” containing “tetris_control” should appear. Create a new tetris_control worker named “tetriscontrol”. It should have three antennas: two (“pull”, (“object”, “bgrid”)) antennas named “blockgrid” and “maingrid”, and a (“push”,”trigger”) antenna “place_init”.
- Connect “blockgrid” to tetriscontrol.blockgrid and “maingrid” to tetriscontrol.maingrid.
- Connect start.outp to tetriscontrol.place_init, but make it the ***second*** connection: after (below) selectblock.select, but before (above) tetrisdraw.start and tetrisdraw.draw.
- Also connect the keyboardsensor to tetriscontrol.place_init. Again, make it the keyboardsensor's ***second*** connection, after selectblock.select but before tetrisdraw.draw.

Save the hivemap and run the hive. Pressing SPACE will now show a block in random orientation at the top middle of the grid.

You can find the completed project of step 7 in the “tetris7/” folder.

Step 8: Falling blocks

In this step, we will make the current block fall, and when it hits bottom, merge it into the maingrid and select a new block.

First, we will adapt the tetris_control worker.
- Open the tetris_control workermap with WorkerGUI.
- Create a trigger output segment (segments.output.push_output, type=trigger) and name it “dropped”. As its “triggerfunc” parameter, specify “trig_dropped”.
- Copy the “dropped” output segment and rename the copy to “lost”. Change its “triggerfunc” parameter to “trig_lost”.
- Select “m_place_init” and append the following two lines to the code field:

```
if self.grid1.overlap(self.grid2):
  self.trig_lost()
```

- Create a “custom import code” worker (segments.custom_code.custom_import_code) and enter “import copy” (no quotes) in its code field.
- Draw a rectangle around “place_init” and “m_place_init” and copy them. Rename the copies to “move_down” and “m_move_down”. Select “m_move_down”, remove the code in the code field and replace it with this:

```
self.get_grids()
block = copy.copy(self.grid2)
block.translate(0,-1)
if block.miny < 0 or self.grid1.overlap(block):
  self.grid1.merge(self.grid2)
  self.trig_dropped()
else:
  self.grid2.translate(0,-1)
```

- Save the workermap. Press Ctrl+G and save the generated code as “workers/tetris_control.py”

Second, we will adapt the tetrismain hivemap.
- Open the tetrismain hivemap with HiveGUI. The “tetriscontrol” worker should have gained a trigger antenna, “move_down”, and two trigger outputs, “dropped” and “lost”.
- Select the keyboardsensor, and delete it.
- Create a float-variable named “period”, and set its value to 0.3
- Create a “cycle” worker (dragonfly.time.cycle). Connect “period” to cycle.period. Connect “cycle” to tetriscontrol.move_down. Then, connect “cycle” to tetrisdraw.draw.
- Connect tetriscontrol.dropped to select_block.select. Then, connect tetriscontrol.dropped to tetriscontrol.place_init. Press the right mouse button a few times while you make the connection to influence its path.
- Create an exit actuator (dragonfly.sys.exitactuator) named “exit”. Connect tetriscontrol.lost to “exit”.

Save the hivemap and run the hive. The blocks will now start to fall and pile up on each other. Once the grid is full (or if you press Esc), the hive will exit.

You can find the completed project of step 8 in the “tetris8/” folder.

Step 9: Moving the block left and right

First, we will adapt the tetris_control worker.
- Open the tetris_control workermap with WorkerGUI.
- Select the “get_grids” node, copy it, and rename the copy to “move_sideways”. Replace the code in the code field with the following:

```
def move_sideways(self, direction):
  self.get_grids()
  block = copy.copy(self.grid2)
  block.translate(direction,0)
  if block.minx < 0: return
  if block.maxx > self.grid1.maxx: return
  if self.grid1.overlap(block): return
  self.grid2.translate(direction,0)
```

- Draw a rectangle around “move_down” and “m_move_down” and copy them. Rename the copies to “move_left” and “m_move_left”. In “m_move_left”, replace the code section with ”self.move_sideways(-1)” (no quotes).
- Draw a rectangle around “move_left” and “m_move_left” and copy them. Rename the copies to “move_right” and “m_move_right”. In the code section of “m_move_right”, change “(-1)” to “(1)”.
- Save the workermap. Press Ctrl+G and save the generated code as “workers/tetris_control.py”

Second, we will adapt the tetrismain hivemap.
- Open the tetrismain hivemap with HiveGUI. The “tetriscontrol” worker should have gained two trigger antennas, “move_left”, and “move_right”.
- Create a keyboard sensor (dragonfly.io.keyboardsensor_trigger) and name it “k_left”. Change its keycode to “LEFT” and connect it to tetriscontrol.move_left.
- Create another keyboard sensor “k_right” with keycode = “RIGHT” and connect it to tetriscontrol.move_right.

- Create a trigger connector (dragonfly.std.pushconnector, type=trigger) named “tetrisdraw_draw” and connect both keyboardsensors to it. Then, connect it to tetrisdraw.draw.

Save the hivemap and run the hive. You can now move the blocks left and right with the arrow keys. The drawing of the grid is updated after each key press.

You can find the completed project of step 9 in the “tetris9/” folder.

Step 10: Rotating and dropping

Here, we will implement clockwise and counter-clockwise rotation of the block, as well as dropping it all the way down.

First, we will adapt the tetris_control worker.
- Open the tetris_control workermap with WorkerGUI.
- Copy “move_sideways” and rename it “rotate”. Change its code field to the following:

```
def rotate(self, times):
  self.get_grids()
  block = copy.copy(self.grid2)
  block.rotate(times)
  if block.minx < 0:
    block.translate(-block.minx,0)
  if block.maxx > self.grid1.maxx:
    block.translate(self.grid1.maxx-block.maxx,0)
  if self.grid1.overlap(block): return
  self.grid2.set(block)
```

- Draw a rectangle around “move_left” and “m_move_left” and copy them. Rename the copies to “rotate_cw” and “m_rotate_cw”. Change the code field of “m_rotate_cw” to “self.rotate(3)”
- Likewise, create “rotate_ccw” and “m_rotate_ccw” with “self.rotate(1)” in its code field.
- Likewise, create “drop” and “m_drop”, with the following code field:

```
self.get_grids()
block = copy.copy(self.grid2)
while block.miny >= 0 and not self.grid1.overlap(block):
  block.translate(0,-1)
block.translate(0,1)
self.grid1.merge(block)
self.trig_dropped()
```

- Save the workermap. Press Ctrl+G and save the generated code as “workers/tetris_control.py”

Second, we will adapt the tetrismain hivemap.
- Open the tetrismain hivemap with HiveGUI. The “tetriscontrol” worker should have gained three trigger antennas: “rotate_ccw”, “rotate_cw”, and “move_right”.
- Create a keyboard sensor (dragonfly.io.keyboardsensor_trigger) and name it “k_down”. Change its keycode to “DOWN”. Connect it to tetriscontrol.drop and then to tetrisdraw_draw.
- Likewise, create two keyboard sensors k_space (keycode=SPACE) and k_return

(keycode=RETURN), connect them to tetriscontrol.rotate_ccw / _cw and then to tetrisdraw_draw.

Save the hivemap and run the hive. Press SPACE or RETURN to rotate the block, and the down arrow key to drop it.

You can find the completed project of step 10 in the “tetris10/” folder.

Step 11: Line removal

Now we will remove full lines from the grid after each block.
- Open the tetris_control workermap with WorkerGUI.
- Select the “get_grids” node, copy it and rename the copy to “remove_lines”. Change the code field to the following:

```
def remove_lines(self):
  values = self.grid1.get_values()
  removed = 0
  y = 0
  while y < self.grid1.maxy+1:
    line = [v for v in values if v[1] == y]
    if len(line) == self.grid1.maxx+1:
      values = [v for v in values if v[1] != y]
      values = [(v[0],v[1]-1) if v[1] > y else v for v in values]
      removed += 1
    else:
      y += 1
  if removed:
    self.grid1.set_values(values)
```

This is probably the most difficult piece of code for Python beginners to understand. First, you select all filled squares (the values). You define y as the *current line position*, starting with the bottom line. As long as the current line position is not above the top line, you select all values of the current line. If the current line is full, you do two things: 1. you filter the list of filled squares, keeping only those *outside* the current line; 2. all filled squares *above* the current line are *moved down* one line. Else, the current line position is lifted one up. In the end, if any lines were removed, you update the grid to contain only the remaining filled squares.

- In the code fields of “m_move_down” and “m_drop”, add a line “self.remove_lines()” in front of “self.trig_dropped()”.
- Save the workermap. Press Ctrl+G and save the generated code as “workers/tetris_control.py”

Save the hivemap and run the hive. Full lines should now be removed as you play.
You can find the completed project of step 11 in the “tetris11/” folder.

Final step: keeping score

Now we will implement scorekeeping, so that points are awarded for every block and full line, and more points if multiple lines are cleared simultaneously.
First, to hold the rewards, we will create a Spyder model TetrisReward. Create a new file

“spydermodels/tetrisreward.spy” and enter the following Spyder code:

```
Type TetrisReward {
  Integer blck
  Integer line
  Integer line2
  Integer line3
  Integer line4
}
```

“blck” indicates the reward for placing a single block, “line” for clearing a single line, and line2-4 for multiple lines. Unfortunately, “block” is a reserved keyword in Spyder.
Save the file, and add the following line to “spydermodels/__init__.py”:

```
from . import tetrisreward
```

Second, we will adapt the tetris_control workermap.
- Open the tetris_control workermap with WorkerGUI
- Create an int-pull buffer named “b_score”. Set its triggerfunc to “get_score”.
- Create an int-pull antenna named “score”. Connect it to “b_score”
- Create an int-push buffer named “b_newscore”. Set its triggerfunc to “trig_newscore”.
- Create an int-push output named “newscore”. Connect “b_newscore” to it.
- Create a pull buffer with type=TetrisReward, named “b_rewards”. Set its triggerfunc to “get_rewards”.
- Create a pull antenna with type=TetrisReward, named “rewards”. Connect it to “b_rewards”.
- In “remove_lines”, add four lines to the end. The new lines are shown in **bold**:

```
  ...
  if removed:
    self.grid1.set_values(values)
    if removed == 1: return self.b_rewards.line
    if removed == 2: return self.b_rewards.line2
    if removed == 3: return self.b_rewards.line3
    if removed == 4: return self.b_rewards.line4
  return 0
```

- In both “m_move_down” and “m_drop”, go to the code field and replace “self.remove_lines()” with the following five lines:

```
self.get_score()
self.get_rewards()
linereward = self.remove_lines()
self.b_newscore = self.b_score + self.b_rewards.blck + linereward
self.trig_newscore()
```

Keep the indentation level of the “self.remove_lines()” command that you replaced.

- Save the workermap. Press Ctrl+G and save the generated code as “workers/tetris_control.py”
Third, we will adapt the tetrismain hivemap.

- Open the tetrismain hivemap with HiveGUI. The “tetriscontrol” worker should have gained a (“pull”, “TetrisReward”) antenna named “rewards”, a (“pull”,”int”) antenna named “score”, and a (“push”,”int”) antenna named “newscore”.
- Connect the “score” variable to tetriscontrol.score. Likewise, connect tetriscontrol.newscore to the “score” variable's input.
- Create a dragonfly.convert.trigger worker, specify its metaparameter “type” as “int”, and name it “update_score”. Connect tetriscontrol.newscore to “update_score”, and “update_score” to tetris_draw_score.draw.
- Finally, specify the rewards. Create a variable with type=TetrisReward, named “rewards”. Edit its value: you can for example set blck to 1, line to 10, line2 to 50, line3 to 100 and line4 to 200. As the very last thing, connect “rewards” to tetriscontrol.rewards.

Save the hivemap and run your hive.

You can find the final completed project in the “final/” folder.

Enjoy!