*Your **submission** for this assignment will have **both a written component and a programming component**. The former **must be a diagram in a single pdf document** with the filename **'comp1501_w18_#########_a3_design.pdf'** and the latter must be a single Python 3 source file with the filename **'comp1501_w18_#########_a3_source.py'.***

*These files **must be compressed into a "zip" file** and uploaded to cuLearn before the deadline.*

***Late assignments will not be accepted** and will **receive a mark of 0**.*

The due date for this assignment is Saturday, March 10, 2018, by 11:00 pm.

For this assignment, you will practice designing and implementing artificial intelligences by creating a module of functions that will govern the behaviour of a combatant in a simulated top-down shooter. To clarify, you have been provided with a program that will handle the updating and rendering of a small arena, and Python's multiprocessing module has been used to allow separate source files to provide the "inputs" that would control each tank. This makes it possible for you to load two finite state machines (each represented as a Python module of Python functions) and observe the ensuing battle!

The largest component of the grade you will receive for this submission will be determined by the performance of your artificial intelligence against each member of a set of artificial intelligences that has also been provided for you. This entails that, if you allow yourself enough time to adequately test your submission, then you will be able to estimate (with a high degree of accuracy) the grade you will receive for your final submission.

Please also note that this assignment will be followed by a tournament where the best artificial intelligences will compete against each another. As a further incentive, the students that submit the best combatants will be given "first-choice" of the possible topics for the final assignment of the semester.

Remember that this program (along with any other program submitted in this class) must be a completely original work, designed and authored by you and you alone, prepared for this offering (i.e., Winter 2018) of COMP1501. Do not discuss this assignment with anyone except the instructor or the teaching assistants, and do not copy anything (source code, algorithm designs, rough work, etc.) from the internet or any other source.

You should approach the tasks required for this assignment in the following order:

1. Familiarize yourself with the "Glad_AI_tors" program by downloading and extracting the contents of the "Source Code for Assignment 3" archive on cuLearn. In order to run this program, the "glad_a_i_tors_v13.py" and "library_for_glad_a_i_tors.py" source files must be in the same folder as the source files for the combatants. The folder labeled "Assets" must appear in this location as well. Start the program from the terminal or the command line by running "glad_a_i_tors_v13.py" and provide the filenames of the two combatants.

2. Observe the battle. Note specifically that the two combatants are placed at random locations, and there are three "pillars" in the center of the map whose positions are also randomized. The first and second artificial intelligence files provided correspond to the red and blue tanks respectively, and beneath each tank are horizontal bars for indicating the weapon power and health of that tank.

3. After your have observed the behaviours of the combatant source files provided, design a finite state machine for an artificial intelligence that you believe will fare well. Please note that this is not an optional design stage; you will be submitting your finite state machine diagram along with the source code for your artificial intelligence, and you will be asked to present your finite state machine diagram before receiving any programming assistance from the teaching assistants or the instructor. See the following page for more details on the capabilities of your tank.

4. Once you have designed your machine and want to evaluate its performance, create a Python 3 source file name "comp1501_w18_#########_a3_source.py", put your name and student number at the top (as comments), and add the following statements:

```
from library_for_glad_a_i_tors import *

def start(arg):
     return ( <string> , <dictionary> )
```

   The return value for every function in your finite state machine should be a tuple. The first element must be a string with the name of the state into which your machine should transition, and the second element will be a dictionary telling your combatant what it should be "doing". If, for example, you would like the finite state machine to begin charging its weapon and then enter the state labeled "wander_around", then the return value should be:

```
return ("wander_around", {'CHARGE': True})
```

Your tank can move quickly around the map and is equipped with both a weapon and a shield. Your tank can move in any direction and aim its weapon independently, and it has no turning radius.

Your weapon is always fired in the direction the gun barrel (which need not be the direction the tank chassis is moving), and if your shot collides with the other tank (n.b., the hitbox of each tank is a circular) then the other tank will receive damage proportional to the "charge" level of the weapon when it was fired.

The weapon must be charged manually and, while charging the weapon, the maximum acceleration and velocity of your tank will be reduced by 30%. When active, the shield will reduce incoming weapon damage by 20% but reduce the maximum acceleration and velocity of your tank by an additional 30%.

Finally, since you cannot pass any "arguments" while transitioning between states, if you have information you would like to retain it can be stored in one of ten "register" variables, labelled 'A' through 'J'.

To summarize the actions available to your tank during transitions, consult the table below.

If you would like your combatant to…	…then add this to the return dictionary
…being charging the weapon…	`'CHARGE': True`
…stop charging the weapon…	`'CHARGE': False`
…activate the shield…	`'SHIELD': True`
…deactivate the shield…	`'SHIELD': False`
…fire the weapon…	`'LAUNCH': True`
…rotate the weapon clockwise…	`'ROT_CW':` *number in (+0, +1)*
…rotate the weapon counter-clockwise…	`'ROT_CC':` *number in (+0, +1)*
…accelerate horizontally…	`'ACLT_X':` *number in (-1, +1)*
…accelerate vertically…	`'ACLT_Y':` *number in (-1, +1)*
…self-destruct…	`'KABOOM': True`
…save value 10 to register variable 'a'…	`'A': 10`

For debugging purposes, please also note that you can write a string of text (e.g., "HELLO") near your tank sprite by adding `'D_TEXT': "HELLO"` to the dictionary, and by adding `'D_LINE': ((x1, y1),(x2, y2))` to the dictionary you can draw a line between two points (x_1, y_1) and (x_2, y_2).

The previous page described "what" your tank can do, so it remains to decide "how" your tank should decide what to do, by providing a function for each state of your finite state machine. The return value for each of these functions (as previously noted) is a tuple of the next state and the current actions, but the only argument passed to each of these functions is an inaccessible variable named "arg". This variable is required by every function in the "library_for_glad_a_i_tors" module and contains the "God's Eye View" of the arena.

You may not, under any circumstances, attempt to access the data contained in this argument. Any attempt to do so will result in an automatic zero for your submission.

For your tank to "sense" the environment, in order to decide how to transition between states, you will need to call some of the following functions. An example of how these functions could be used is detailed on the next page.

`get_position(arg) / get_velocity(arg)`

returns the current position (x, y) / velocity (dx, dy) of your tank

`get_n_border(arg) / get_s_border(arg)`

returns the y-position of the north / south border

`get_e_border(arg)/get_w_border(arg)`

returns the x-position of the east / west border

`get_weapon_angle(arg)`

returns the current weapon angle (in degrees)

`get_weapon_power(arg)`

returns the current weapon power (as an integer)

`get_damage_level(arg)`

returns the amount of damage your tank has received

`def is_charge_active(arg) / is_shield_active(arg)`

returns a Boolean to indicate if the tank is charging / the shield is active

`def get_saved_data(arg, register_name)`

returns the value stored in the register variable named in the second argument

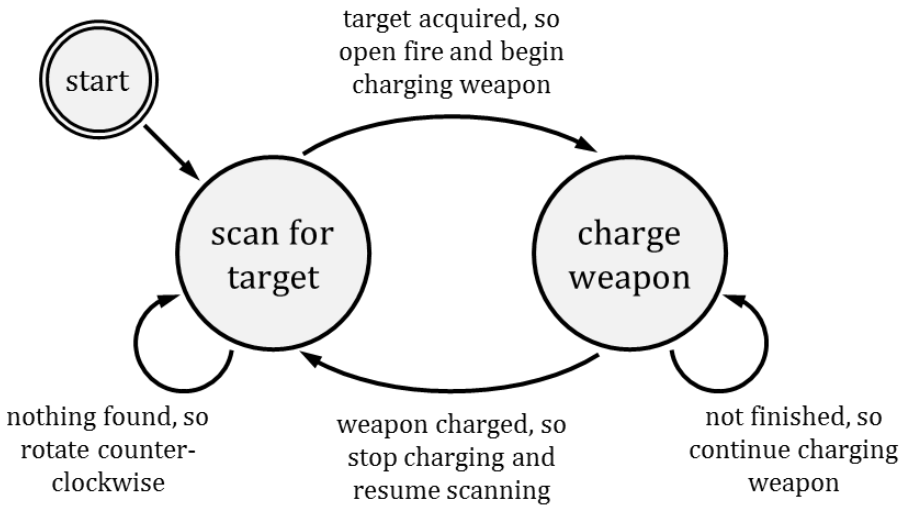
`def get_radar_data(arg, radar_angle)`

returns a tuple of the type (either "opponent" or "obstacle") and distance of the nearest object to the tank, in whatever direction is passed as the second argument

To help with your understanding of the design and implementation of a "Glad_AI_tors" artificial intelligence, consider the example of a stationary (and rudimentary) opponent:

This tank rotates its weapon in a counter-clockwise direction, repeatedly "scanning" in whatever direction the weapon is pointing. If the scan ever reveals the presence of an enemy, the weapon is fired. Once the weapon has been fired, the tank starts recharging the weapon, and then when the weapon power has reached 30%, the search for the opponent resumes.

Here is the finite state machine for the artificial intelligence that has been described above:



Here is the "Glad_AI_tors" source code that corresponds to the finite state machine above:

```
from library_for_glad_a_i_tors import *

def start(arg):
  return ("scan_for_target", {'CHARGE': True})

def scan_for_target(arg):
  current_weapon_angle = get_weapon_angle(arg)
  (detected, _) = get_radar_data(arg, current_weapon_angle)
  if detected == "opponent":
       return ("charge_weapon", {'LAUNCH': True, 'CHARGE': True, 'ROT_CC': 0})
  else:
       return ("scan_for_target", {'ROT_CC': 1})

def charge_weapon(arg):
  if get_weapon_power(arg) > 30:
       return ("scan_for_target", {'CHARGE': False})
  else:
       return ("charge_weapon", {'CHARGE': True})
```