# *Lab 9 – State Machine Diagrams*

Date assigned: Monday, March 13, 2016

Date due: **Monday, March 13, 2016, 14:50**

**Learning Objectives**

Upon successful completion of this lab exercise, the student will be able to:

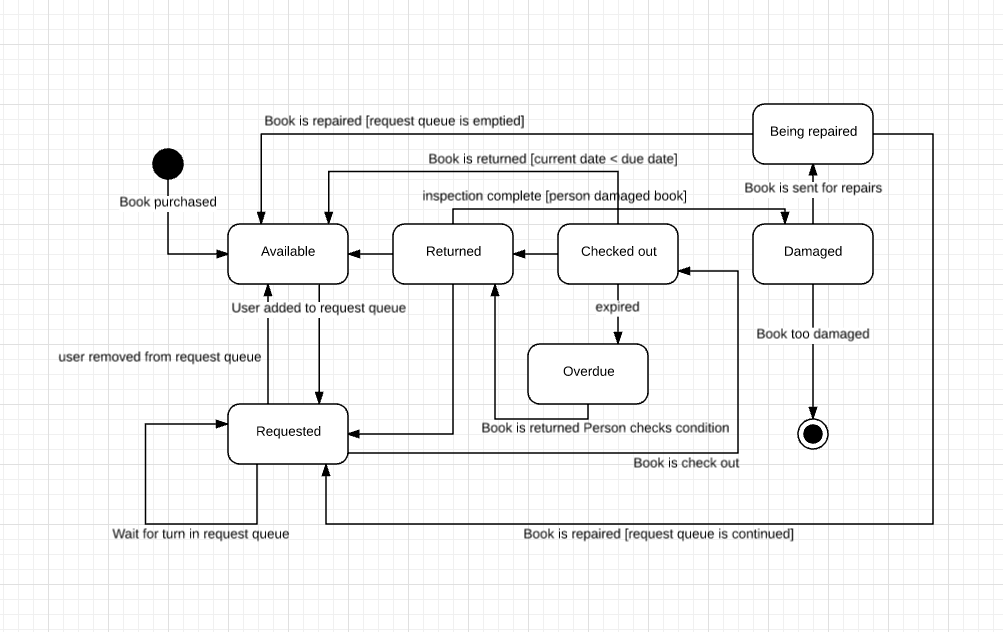
* Practice with State Machine Diagrams

Lab Set Up

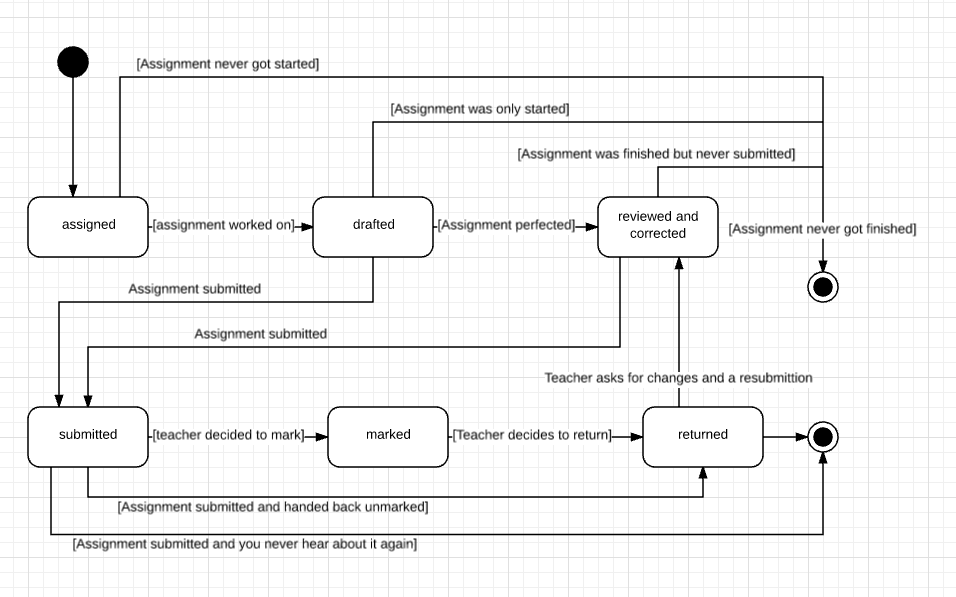
1. Use Lucidchart to create diagrams and paste them in this document. Rename it to YourUserName\_E21\_L09\_State\_Machine\_Diagrams.docx and submit to Moodle by end of lab.

To do:

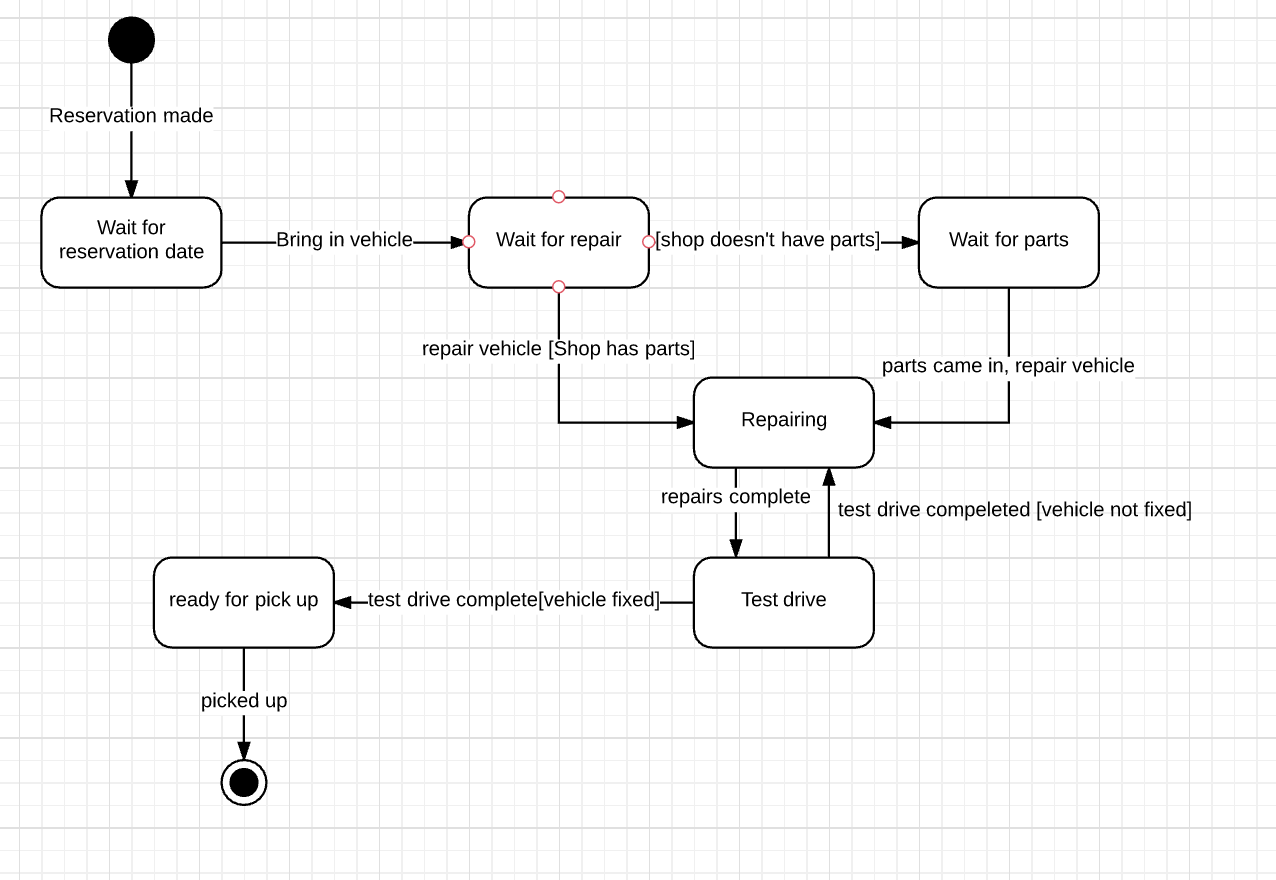
1. Draw the behavioral state machine for the life cycle of an instance of a Book class in a library. The states identified for a book object include Available, Checked Out, Overdue, Requested, Damaged, and Being Repaired. Certain transitions probably do not apply; for instance, it makes no sense to have a book to go from a repaired state to a damaged state. However, going from a damaged state to a repaired state makes sense. Nor does it make sense for a book to go from an available state directly to an overdue state. However, the converse makes sense. Note that a book might be too damaged; in which case we transition to the final pseudo state.



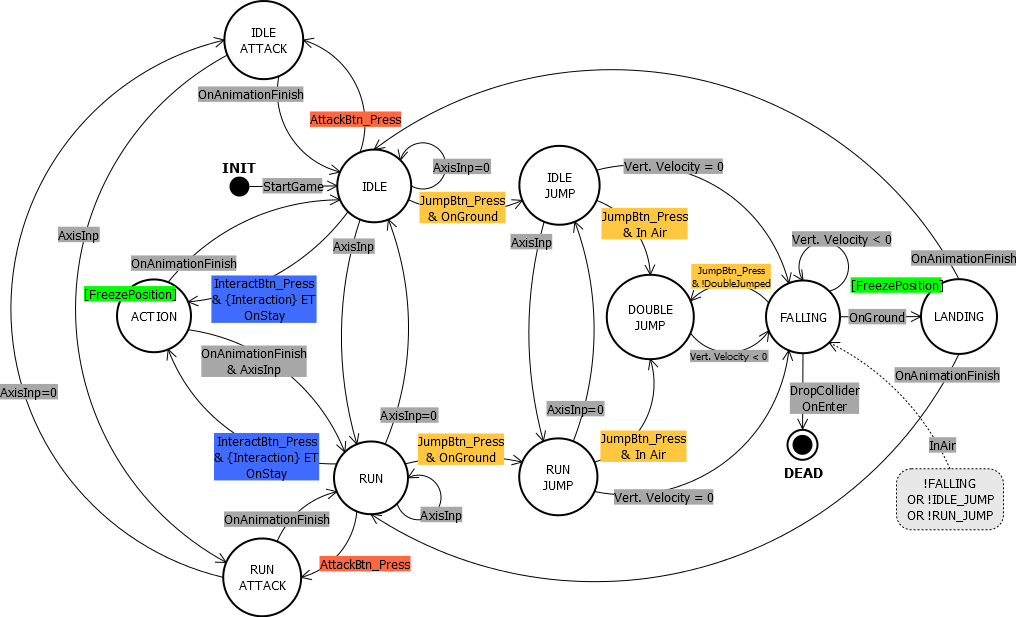
1. Draw a state machine diagram for the states that an assignment goes through from the time it is assigned by the instructor, to a draft being completed, the draft assignment being reviewed until it is correct, the assignment being submitted, marked and returned. Remember to use triggers and behaviours for the transitions between the states and guard conditions for when the transitions will take place.



1. Reservations are made for repairs of vehicles. The vehicle is delivered and waits in the dock until repair can commence. Repair can commence at any time. During repair new parts may be needed and the vehicle may have to wait for the parts to arrive. When repairs are completed a test drive is organized. Following the test drive, if the vehicle does not pass the test drive, more repairs are needed. If not more repairs the vehicle will become ready to be picked up. Two days later the vehicle is picked up by its owner.



1. Early video games extensively used state machines such as this one for a platformer:



Read [this article](http://gameprogrammingpatterns.com/state.html). (You can stop when you get to the section “The State Pattern”. The rest is just implementation details.)

Answer the following:

Define FSM, and what is meant by “finite” and “state”?

A finite state machine is one which can only be in one of a finite number of different states during it’s execution. For example, you’re building a lexer for a compiler and need to build up an array of different tokens for the lexer. When an a string is encountered for example, you know that now while in the state of string, you need to ignore all potential operators or key words. Or when you’re in the state of an arithmetic expression, you need to treat certain operators differently. So two numbers will be added instead of appended as they would if they were strings. These different states help determine what kinds of tokens will be built. In most cases, there will be a very limited number of different states that a machine can be in though. In a lexer, you’d have String, Number, Operator, etc.

What do the nodes of a FSM diagram represent?

The different nodes represent the different states that the machine can be in at a given time. In the state machine for the game above, it can be in 10 different states, and has specific paths that it can follow to move between different states.

What to the connections between the nodes represent?

The connections represent the different paths between different states that a machine can follow. For example, this machine can go from a run, to a run attack. This is done by the event of AttackBtn\_Press event getting fired. So when in the run state, if the AttackBtn\_Press event is fired, the state will change to run attack. However, you cannot move from the run attack state directly to a double jump. You need to go through a series of different states before you can get there.

Examine the code at the top of the article (with bugs) and the example provided in [Enum and Switches](http://gameprogrammingpatterns.com/state.html#enums-and-switches). What is the benefit to going to the state-driven model? What would be easier to read in order to understand the game, the code (enum/switch version) or the State machine diagrams?

Using the state machine with enums and switch statements is far easier to read than using the chunk of if statements. Flags are good when you have only a few of them and they’re simple things, like whether or not your item was found. But when you’re dealing with something larger than that and you can only have one state active at a time, then enums with switch statements are much clearer, much more efficient and much more robust. There’s no forgetting to set a flag and failing to maintain proper state.

1. Assessment
2. What did you learn in completing this lab?

I learned about state machines, which are awesome.

1. What did you have difficulty with?

A lack of understanding of the syntax made specific thinking much more difficult

1. What did you do well?

Learned state machines

1. How many hours did you spend in completing this lab?

3?

1. What took you the most time?

Waiting for help/marking

Mark breakdown

|  |  |
| --- | --- |
| **Part A** |  |
| Book FSM | 10 |
| **Part B** |  |
| Assignment FSM | 10 |
| **Part C** |  |
| Reservation FSM | 10 |
| **Part D** |  |
| Games FSM Analysis | 5 |
| Self-assessment and properly handed in, English | 5 |

To Submit

Copy the following files to the Moodle directory for this course:

* YourUserName\_E21\_L09\_State\_Machine\_Diagrams.docx