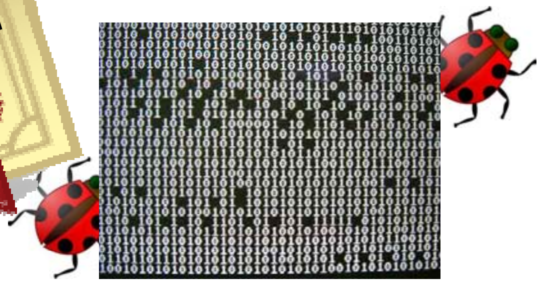
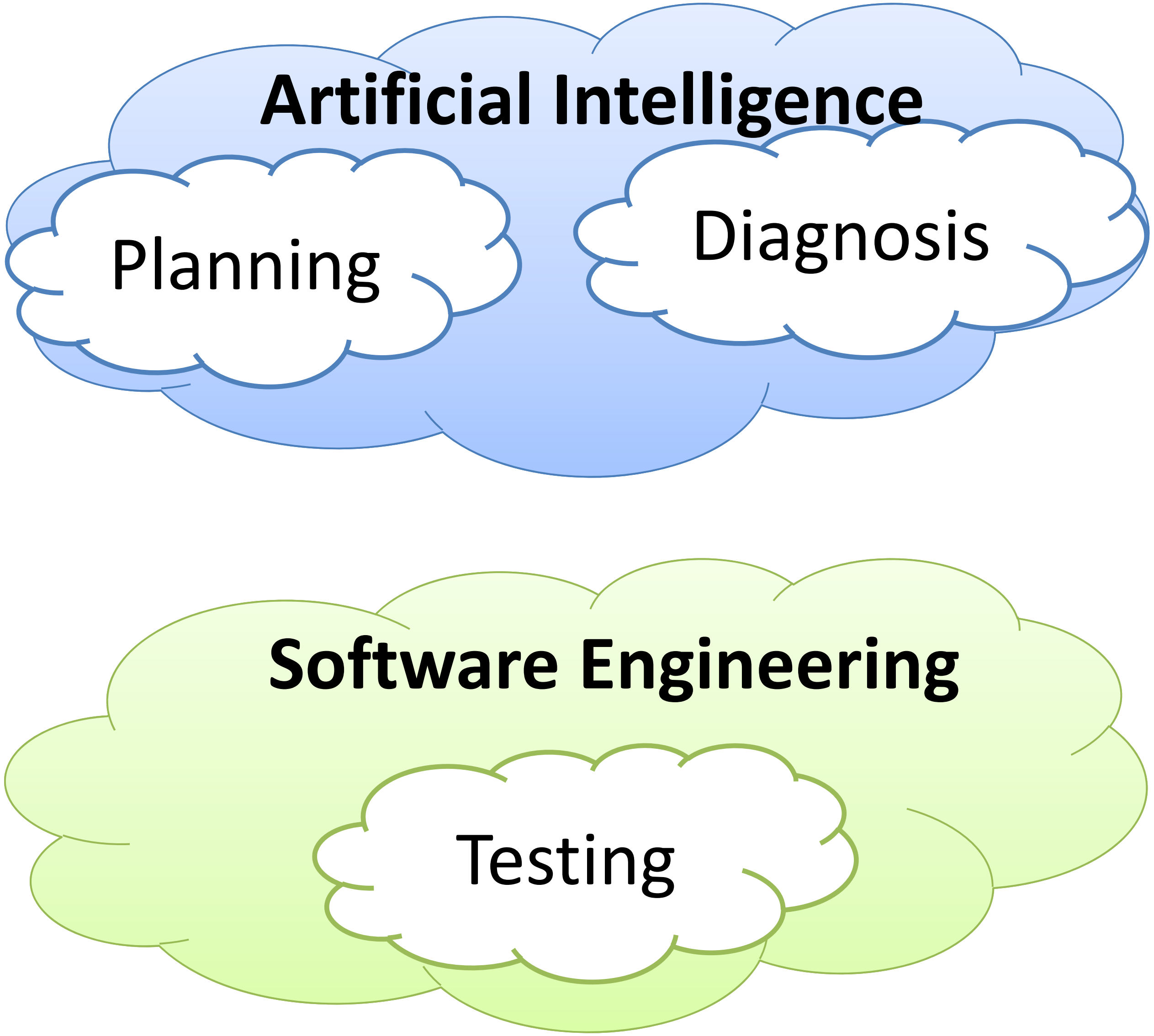
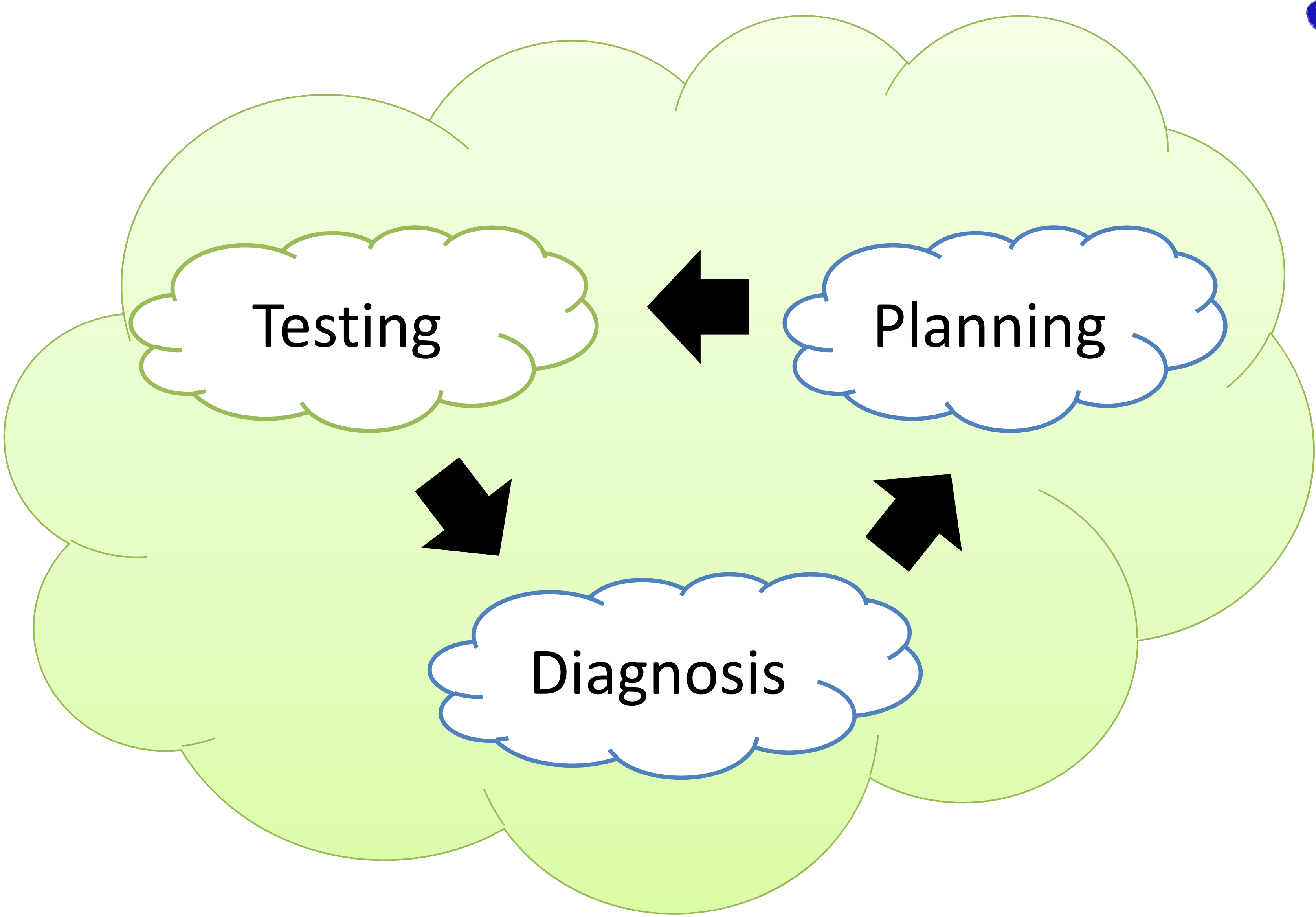


Integrating Artificial Intelligence
in
Software Testing

Roni Stern and **Meir Kalech**, ISE department, BGU

Niv Gafni, **Yair Ofir** and **Eliav Ben-Zaken**, Software Eng., BGU


Artificial Intelligence
Planning
Diagnosis
Software Engineering
Testing


Testing
Planning
Diagnosis

Testing is Important

- Quite a few software development paradigms
- All of them recognize **the importance of testing**
- There are several types of testing
  - E.g., unit testing, **black-box (functional) testing** ….

≠

Programmer

- Write programs
- Follows spec.
- Goal: fix bugs!

Tester

- Runs tests
- Follows a test plan
- Goal: find bugs!

Handling a Bug

1. Bugs are reported by the tester
   - "Screen A crashed on step 13 of test suite 4…"
2. Prioritized bugs are assigned to programmers
3. **Bugs are diagnosed**
   - What caused the bug?
4. **Bugs are fixed**
   - Hopefully…

DEBUG!

Why is Debugging Hard?

- The developer needs to **reproduce the bug**
- Reproducing (correctly) a bug is non-trivial
  - **Bug reports may be inaccurate** or missing
  - Software may have **stochastic elements**
    - Bug occurs only once in a while
  - Bugs may appear only in **special cases**

Let the tester provide more information!

Introducing AI!

Tester
- Run a test suite
- Find a bug

→

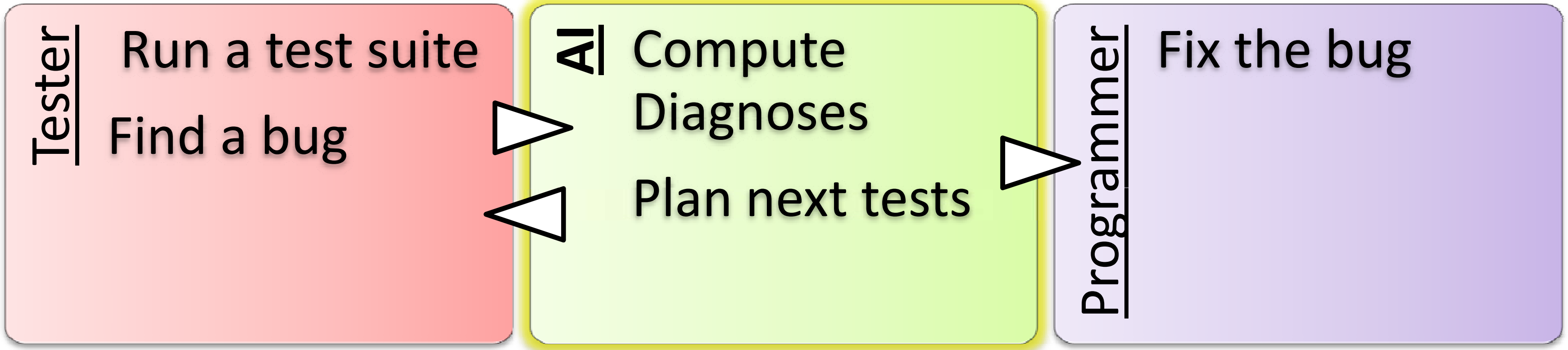
Programmer
- Diagnose
- Fix the bug

Tester
- Run a test suite
- Find a bug

AI
- Compute Diagnoses
- Plan next tests

Programmer
- Fix the bug

Test, Diagnose and Plan (TDP)



1. The tester runs a set of planned tests (test suite)
2. The tester finds a bug
3. AI **generates possible diagnoses**
4. If there is only one candidate – pass to programmer
5. Else, AI **plans new tests** to prune candidates

Diagnosis

Find the **reason** of a problem

given observed **symptoms**

Requirement: **knowledge** of the diagnosed system

- Given by experts
- Learned by AI techniques

Model-Based Diagnosis

- Given: a **model of how the system works**

➔Infer what went wrong

- Example:

```
IF ok(battery) AND ok(ignition)
THEN start(engine)
```

- What if the engine doesn't start?

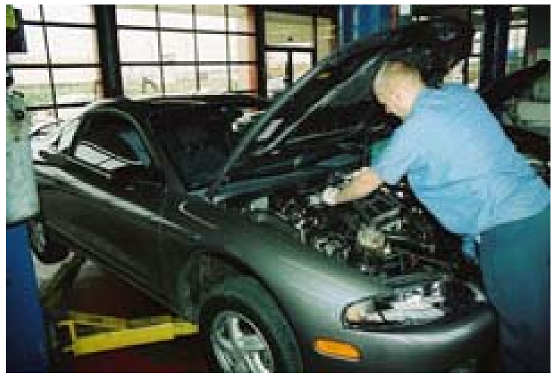
Where is MBD applied?

- Printers

  (Kuhn and de Kleer 2008)

- Vehicles

  (Pernestål et. al. 2006)



- Robotics

  (Steinbauer 2009)

- Spacecraft - Livingstone

  (Williams and Nayak, 1996; Bernard *et al., 1998)*

....

Software is Difficult to Model

- Need to code how the software should behave
  - **Specs are complicated to model**
- Static code analysis **ignores runtime**
  - Polymorphism
  - Instrumentation

…

Zoltar [Abrue et. al. 2011']

- Construct a model from the observations
- Observations should include execution traces
  - Functions visited during execution
  - Observed outcome: bug / no bug
- Weaker model

Ok(*function1*) → *function1* outputs valid value

➔ A bug entails that at least one comp was not Ok

Execution Matrix

- A key component in Zoltar is the **execution matrix**
- It is built from the observed execution traces
  - Observation 1 (BUG) : F1→F5→F6
  - Observation 2 (BUG) : F2→F5
  - Observation 3 (OK) : F2→F6→F7→F8

	F1	F2	F3	F4	F5	F6	F7	F8	Bug
Obs1	1	0	0	0	1	1	0	0	1
Obs2	0	1	0	0	1	0	0	0	1
Obs3	0	1	0	0	0	1	1	1	0

Diagnosis = Hitting Sets of Conflicts

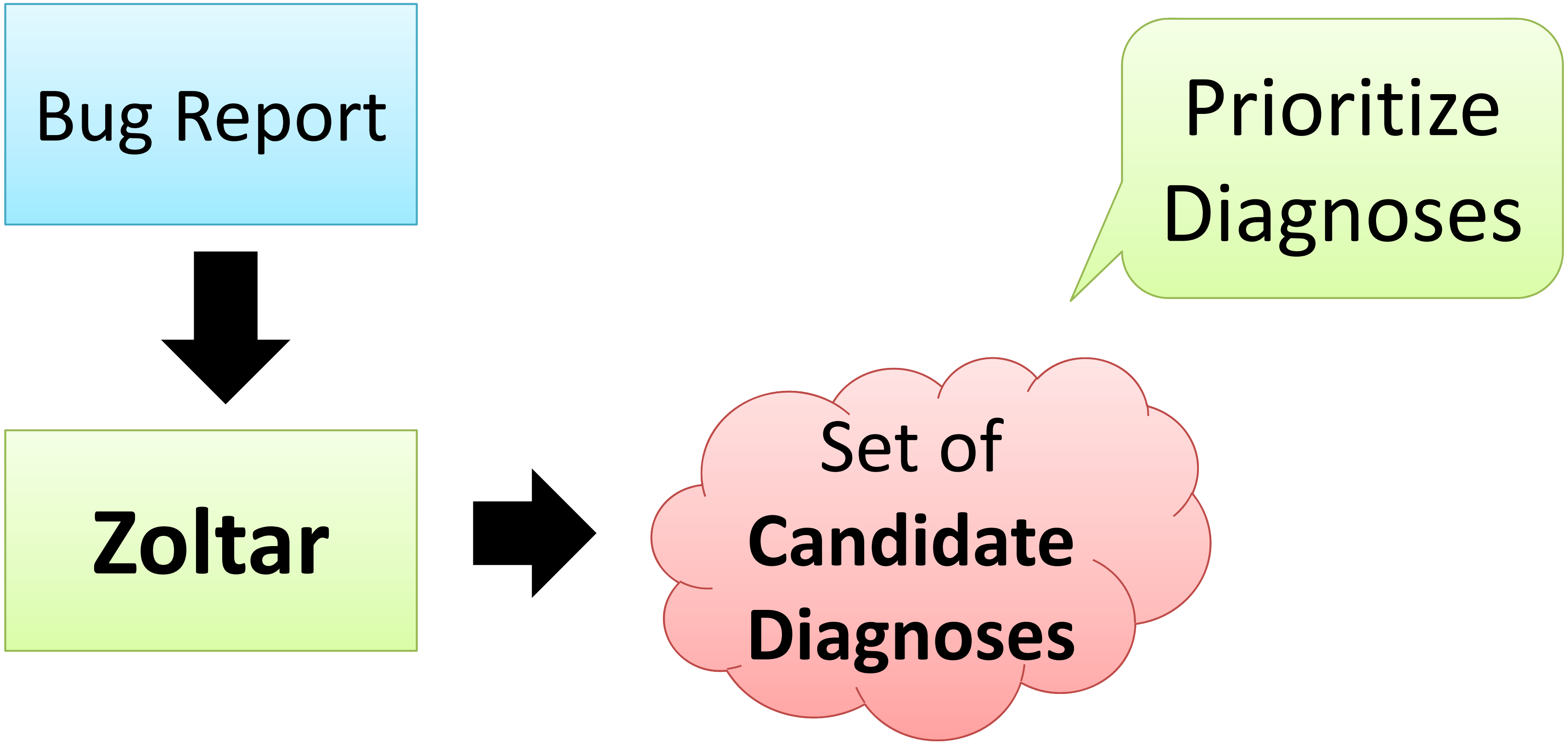
In every BUG trace at least one function is faulty

- Observations:
  - Observation 1 (BUG) : F1→F5→F6
  - Observation 2 (BUG) : F2→F5
- **Conflicts**:
  - Ok(**F1**) AND Ok(**F5**) AND Ok(**F6**)
  - Ok(**F2**) AND Ok(**F5**)

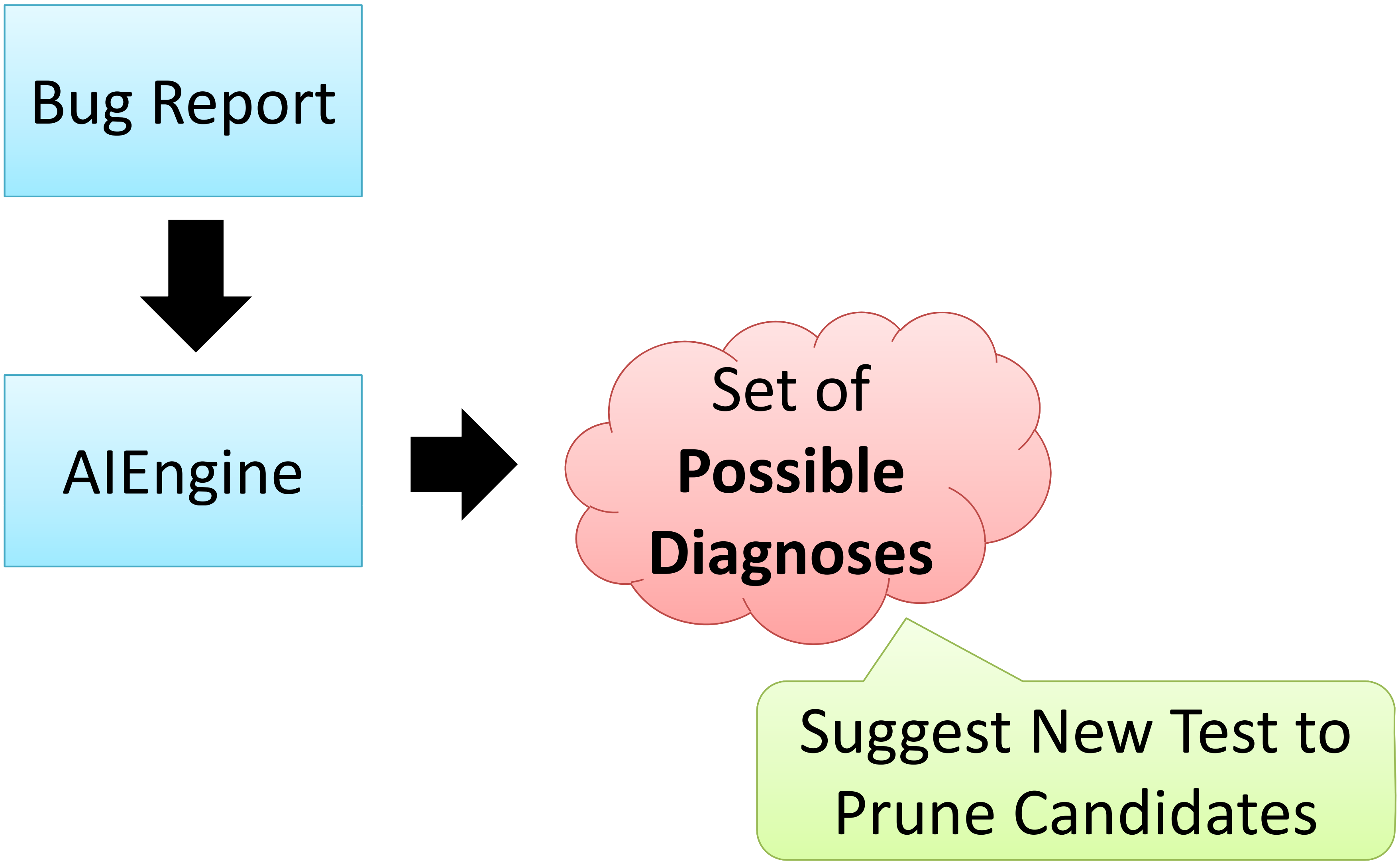
Hitting sets of Conflicts

- **Possible diagnoses:** {**F5**} ,{**F1**,**F2**},{**F6**,**F2**}

Software Diagnosis with Zoltar



Plan More Tests



Plan More Tests

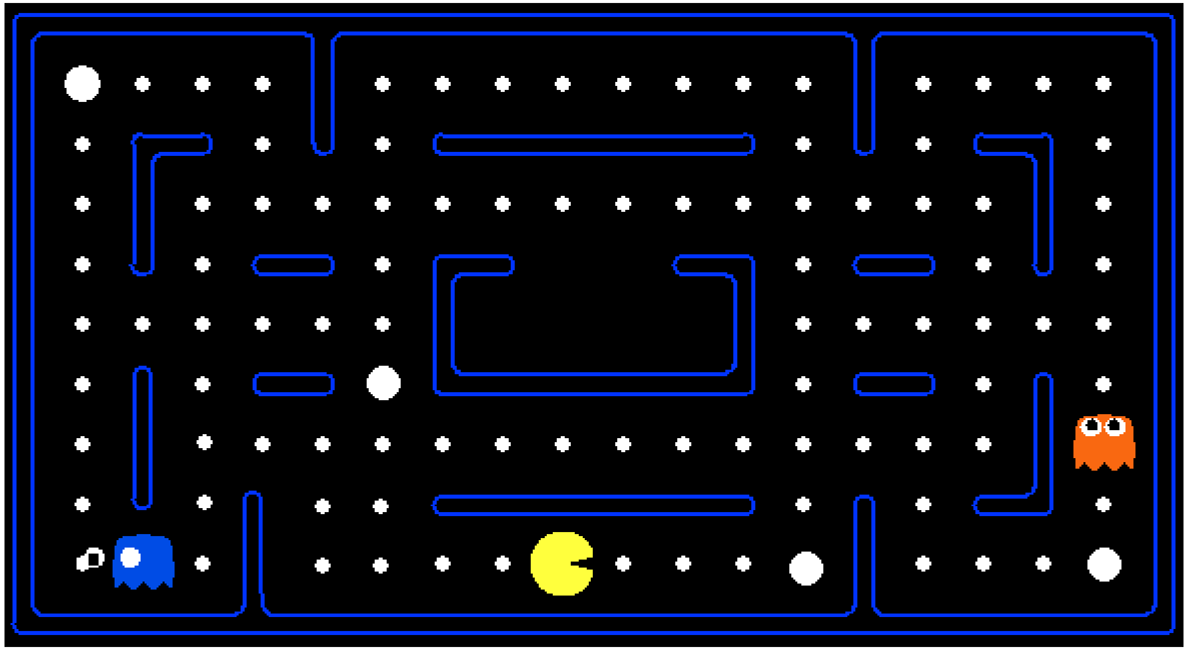
Bug Report

AIEngine

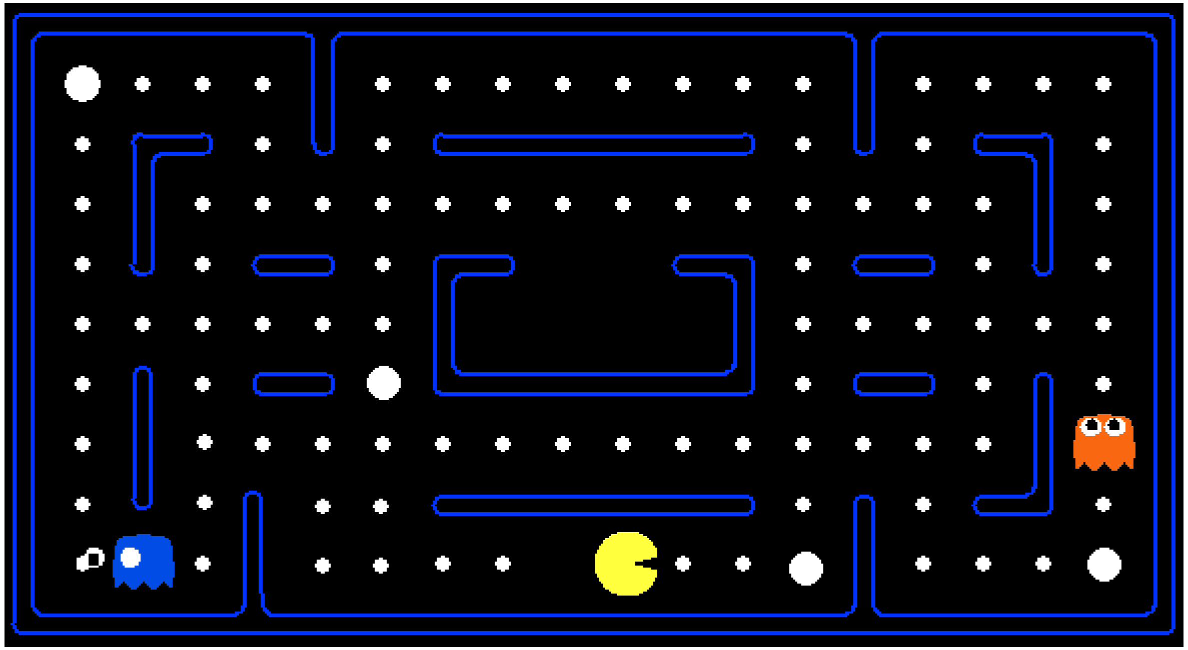
A single diagnosis

Suggest New Test to Prune Candidates

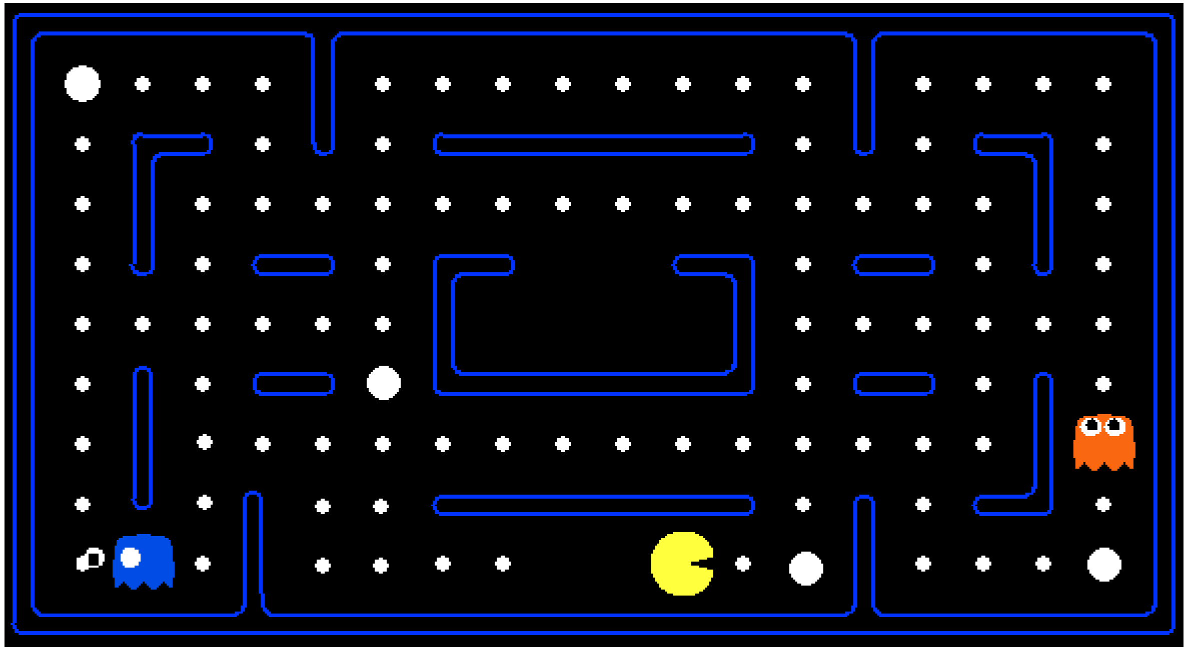
Pacman Example



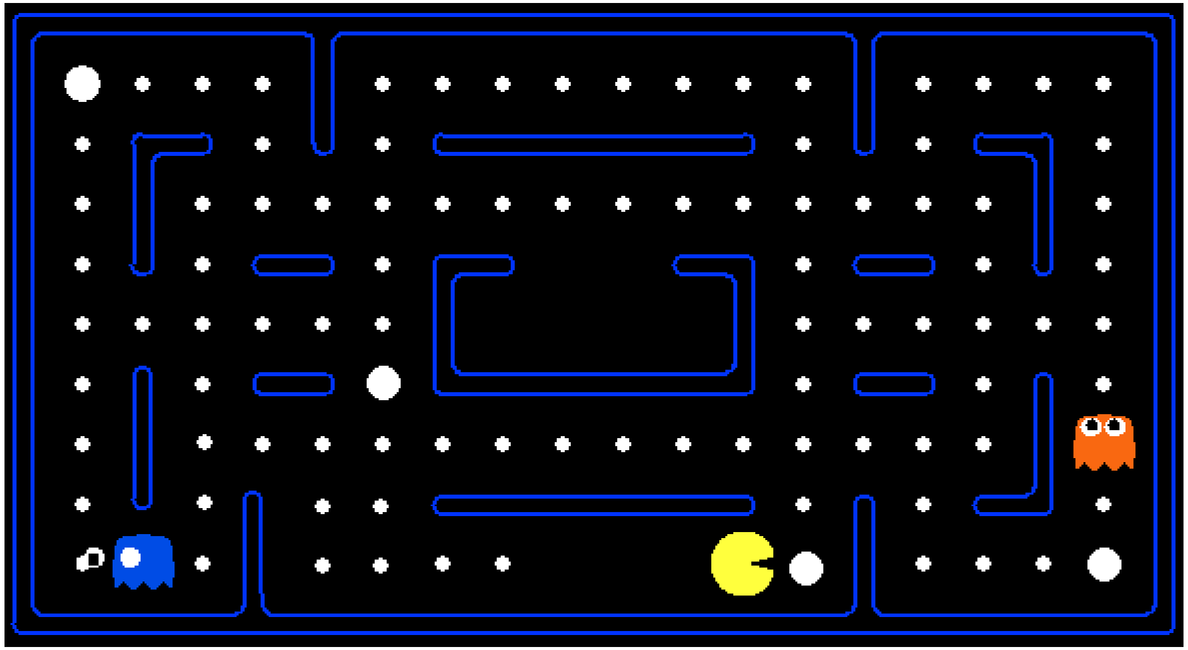
Pacman Example



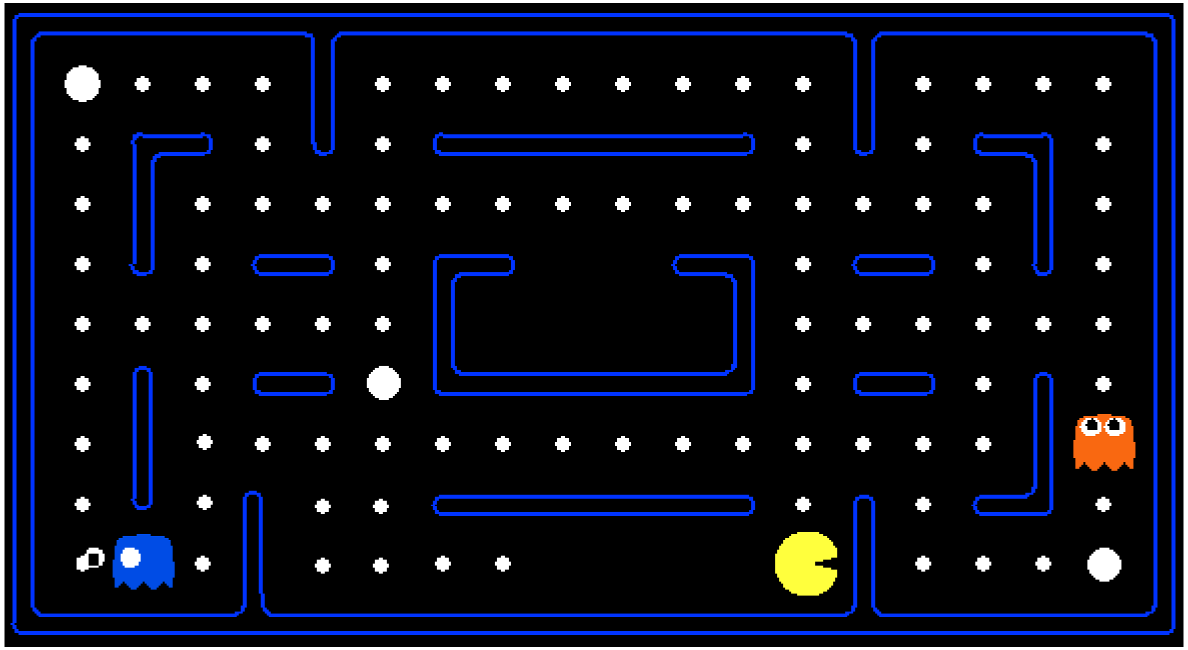
Pacman Example



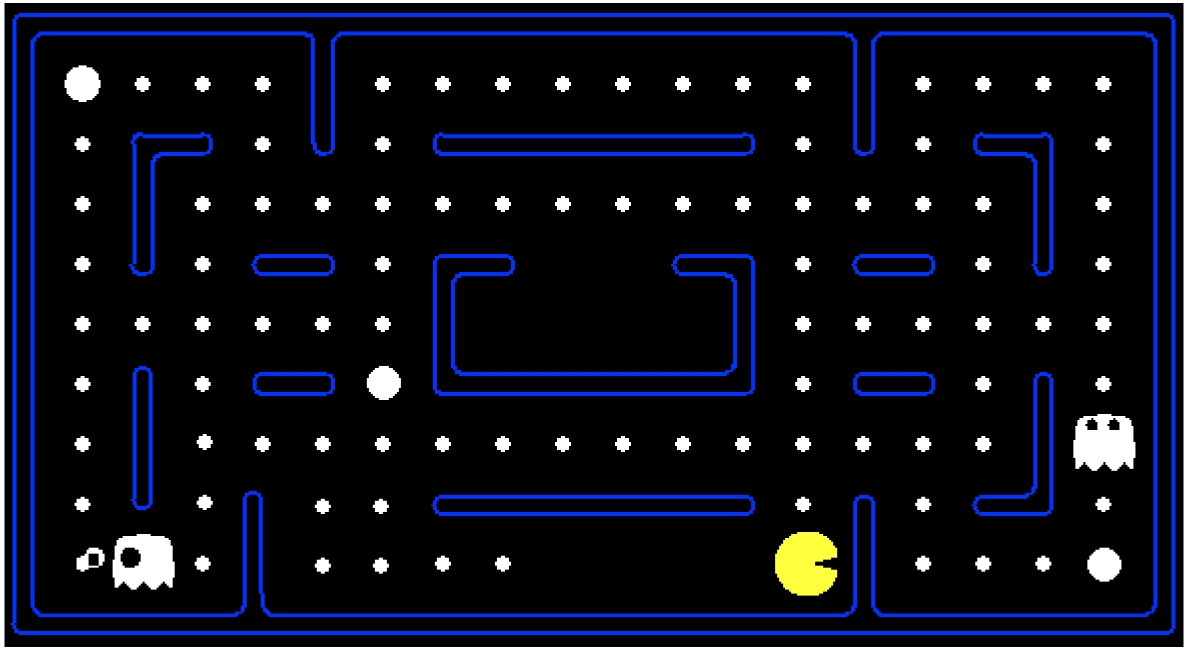
Pacman Example



Pacman Example



Pacman Example

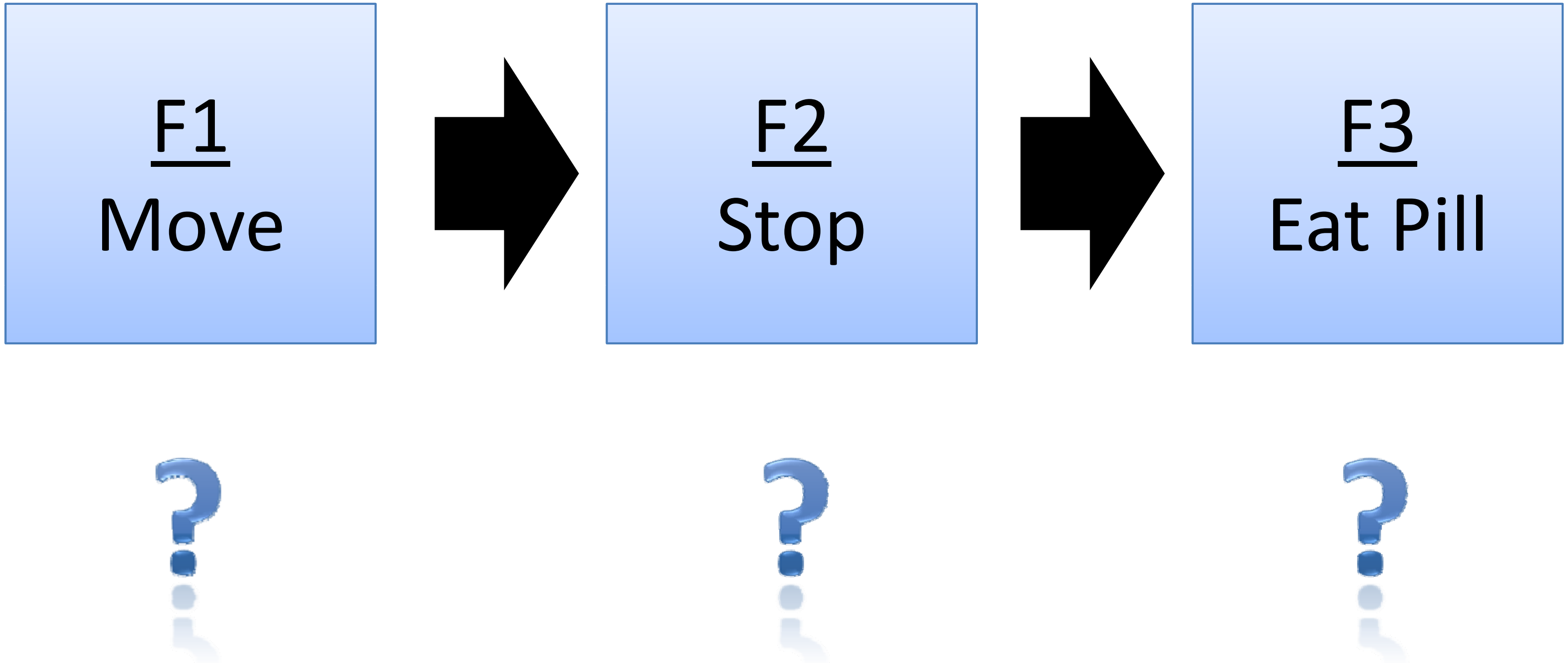


Pacman Example

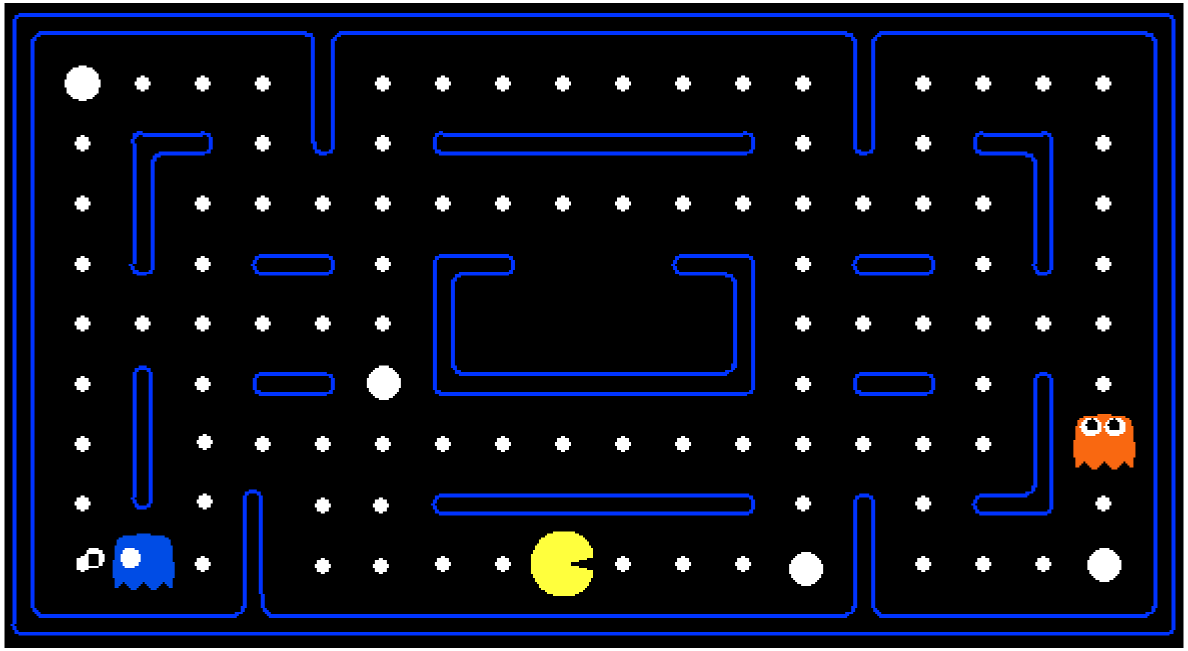
C:\Windows\system32\cmd.exe

```
Win Rate:      0/1 (0.00)
Record:        Loss

D:\projects\search>python pacman.py
Pacman died! Score: -537
Average Score: -537.0
Scores:        -537
Win Rate:      0/1 (0.00)
Record:        Loss

D:\projects\search>python pacman.py
Pacman died! Score: -535
Average Score: -535.0
Scores:        -535
Win Rate:      0/1 (0.00)
Record:        Loss

D:\projects\search>python pacman.py
Pacman died! Score: -525
Average Score: -525.0
Scores:        -525
Win Rate:      0/1 (0.00)
Record:        Loss

D:\projects\search>_
```

Execution Trace

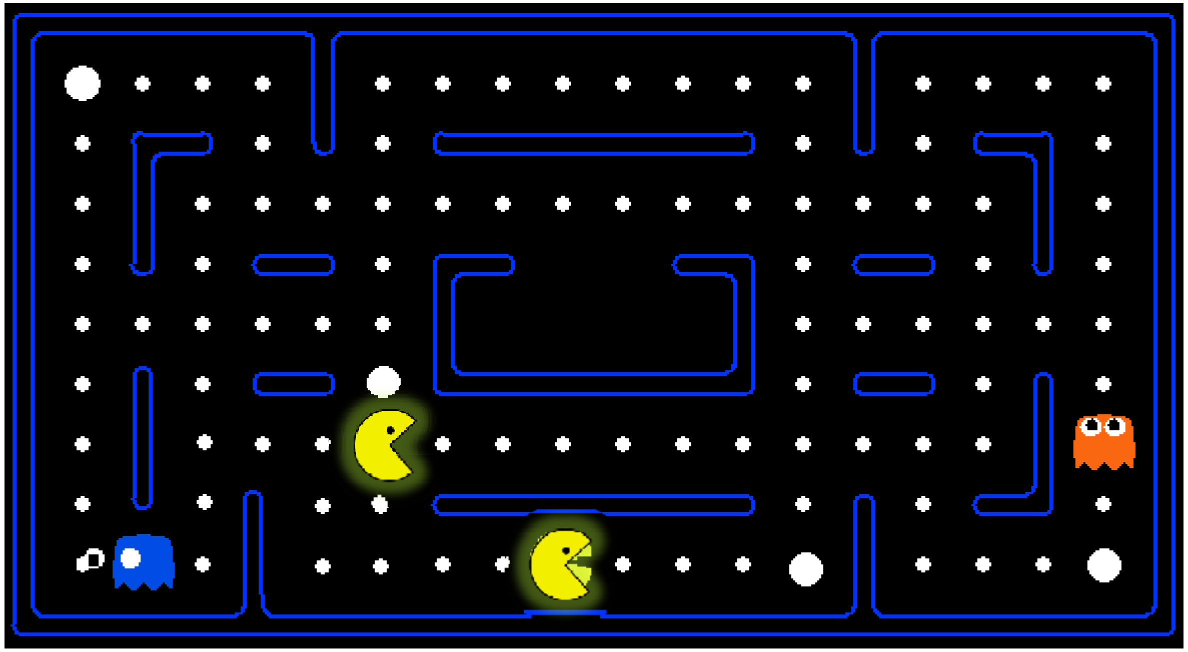


Which Diagnosis is Correct?



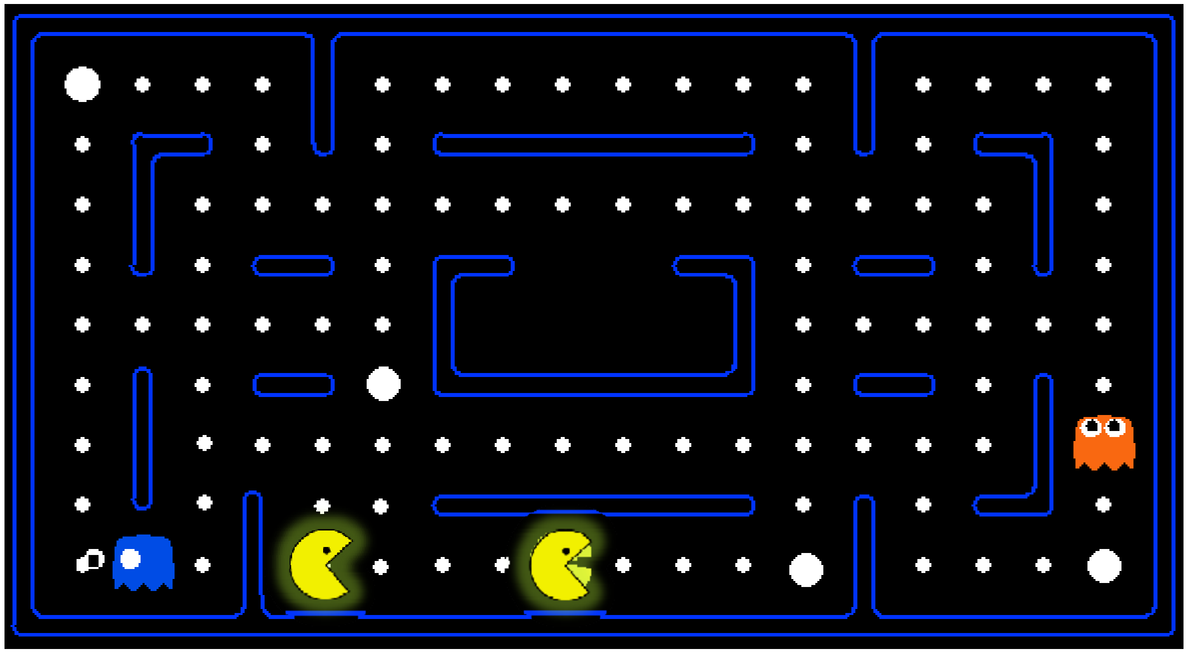
Knowledge

- Most probable cause: Eat Pill



Knowledge

- Most probable cause: Eat Pill
- Most easy to check: Stop



Objective

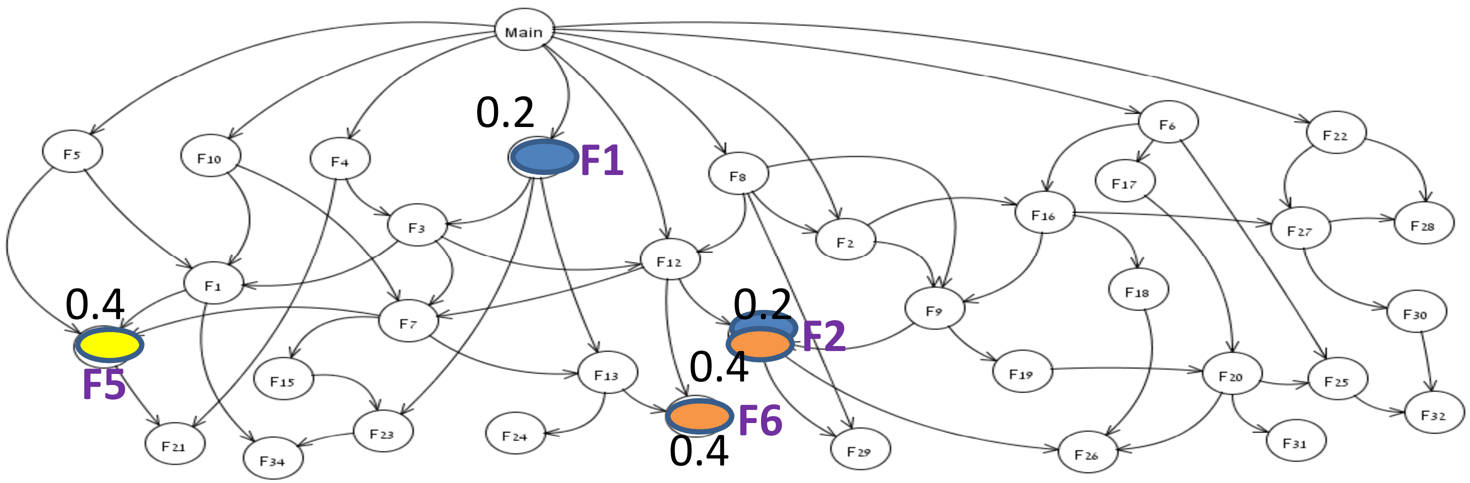
Plan a sequence of **tests**

to find the **best diagnosis**

with **minimal tester actions**

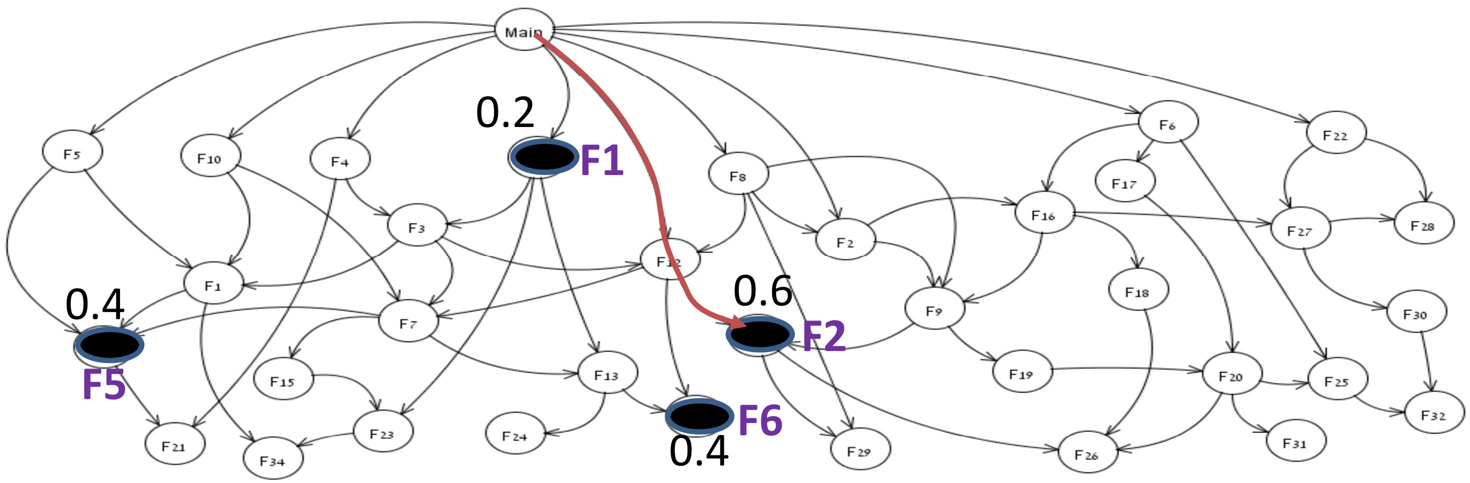
1. Highest Probability (HP)

- Compute the prob. of every function $\mathbf{C_i}$
  = Sum of prob. of candidates containing $\mathbf{C_i}$
- Plan a test to **check the most probable function**



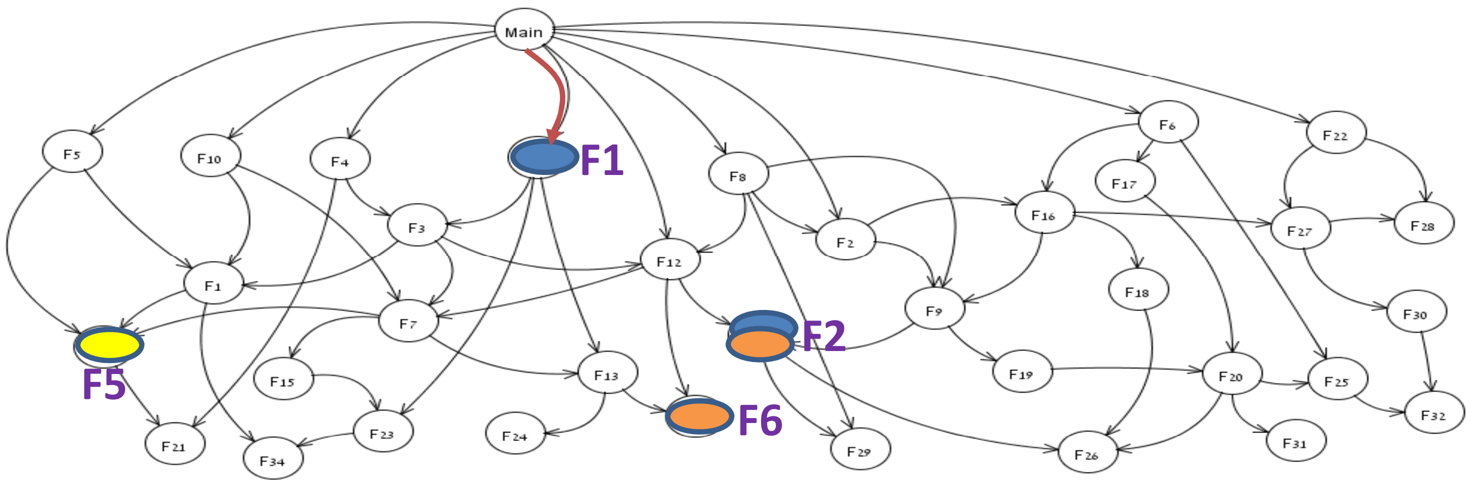
1. Highest Probability (HP)

- Compute the prob. of every function $\mathbf{C_i}$
  = Sum of prob. of candidates containing $\mathbf{C_i}$
- Plan a test to **check the most probable function**



2. Lowest Cost (LC)

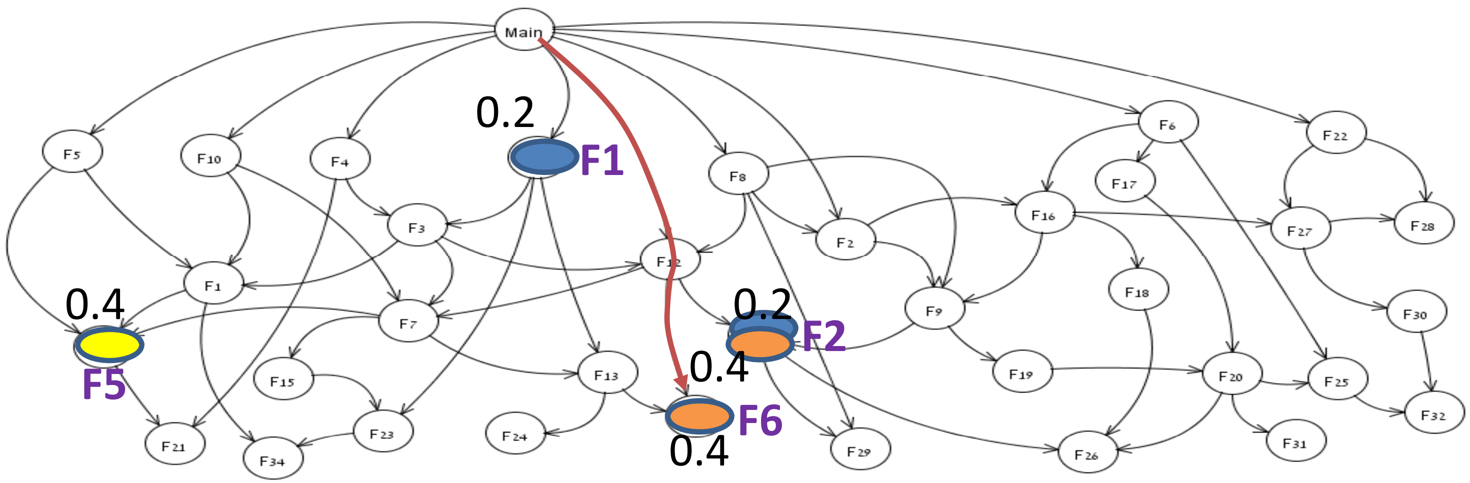
- Consider only functions C_i with **$0<P(C_i)<1$**
- **Test the closest function** from this set



3. Entropy-based

- A test **α** = a set of components
  - $\mathbf{\Omega_+}$ = set of candidates that assume **α** will **pass**
  - $\mathbf{\Omega_-}$ = set of candidates that assume **α** will **fail**
  - $\mathbf{\Omega_?}$ = set of candidates that are **indifferent** to **α**
- Prob. **α** will pass = prob. of candidates in Ω_+
- Choose **test with lowest Entropy(Ω_+ ,Ω_- ,$\Omega_?$)**

  = highest information gain



4. Planning Under Uncertainty

- Outcome of test is unknown

  → we would like to **minimize expected cost**
- Formulate as Markov Decision Process (MDP)
  - **States**: set of performed tests and their outcomes
  - **Actions**: Possible tests
  - **Transition**: $Pr(\mathbf{\Omega}_{+})$, $Pr(\mathbf{\Omega}_{-})$ and $Pr(\mathbf{\Omega}_{?})$
  - **Reward** (cost): Cost of performed tests
- Current solver uses myopic sampling

Preliminary Experimental Results

- Synthetic code
- Setup
  - Generated random call graphs with 300 nodes
  - Generate code from the random call graphs
  - Injected 10/20 random bugs
  - Generate 15 random initial tests
- Run until a diagnosis with prob.>0.9 is found
  - Results on 100 instances

Preliminary Experimental Results

	MDP		LC		Entropy		HP	
Prob.	10	20	10	20	10	20	10	20
0.1	7.5	10.1	11.0	10.6	24.3	63.0	29.4	87.4
0.2	13.2	15.7	16.9	20.4	39.3	92.7	38.5	115.7
0.3	15.7	19.4	20.2	24.8	43.2	99.4	44.7	118.8
0.4	17.2	22.1	22.6	30.0	49.2	103.3	52.7	120.7
0.5	20.1	24.9	27.5	35.8	56.2	107.0	71.3	130.8
0.6	23.8	25.6	31.1	37.7	57.8	111.0	71.8	132.7
0.7	25.8	26.8	31.2	38.7	65.1	111.9	74.3	133.9
0.8	26.0	32.2	32.4	39.9	69.9	115.3	77.6	135.3
0.9	26.9	33.0	35.0	42.2	73.0	143.9	79.0	136.4

- HP and Entropy perform worse than (LC) and MDP
- probability based (HP and Entropy) perform worse than the cost based approach.
- MDP which considers cost and probability outperforms the others
- 20 bugs is more costly for all algorithms than 10 bugs

Preliminary Experimental Results - NUnits

- Real code
- Well-known testing framework for C#
- Setup
  - Generated call graphs with 302 nodes from NUnits
  - Injected 10,15,20,25,30 random bugs
  - Generate 15 random initial tests
- Run until a diagnosis with prob.>0.9 is found
  - Results on 110 instances

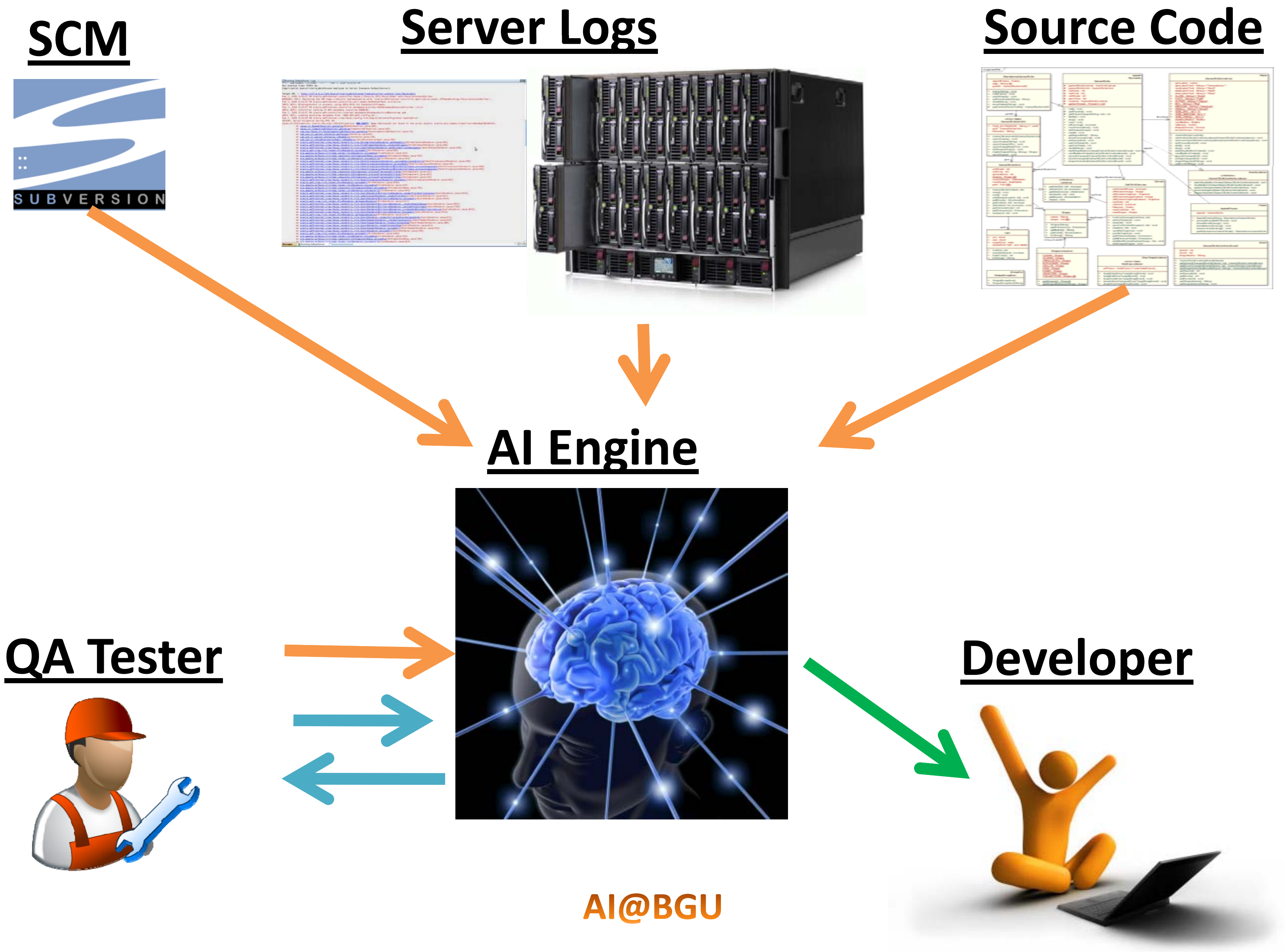
Preliminary Experimental Results - NUnits

Prob.	MDP	LC	Entropy	HP
0.1	2.34	1.92	4.00	5.15
0.2	3.16	2.50	5.79	7.06
0.3	4.18	3.54	15.42	17.47
0.4	4.88	7.00	50.70	66.58
0.5	5.08	7.00	50.75	66.75
0.6	6.45	8.06	53.85	70.14
0.7	6.45	8.06	53.85	70.14
0.8	6.45	8.09	53.85	70.14
0.9	6.45	8.17	55.69	70.41

- Similar results
- The cost increases with the probability bound
- **MDP which considers cost and probability outperforms the others**

Conclusion

- The test, diagnose and plan (TDP) paradigm:
  - AI **diagnoses observed bug** with MBD algorithm
  - AI **plans further tests** to find correct diagnosis
- Empowers tester using AI
  - Bug diagnosis done by tester+AI


SCM
SUBVERSION
Server Logs
Source Code
AI Engine
QA Tester
Developer
AI@BGU