GDC

Where The $@*&% Are Your Tests?!

Kevin Dill
Senior Solutions Engineer
kevind@kythera.ai

KYTHERA AI

Background & Agenda

KYTHERA AI

- AI developer for 20+ years
- "Unit test curious" for ~15 years…
- But unit testing just didn't seem to work well for games!
- Around 6 or 7 years ago, it finally "clicked"

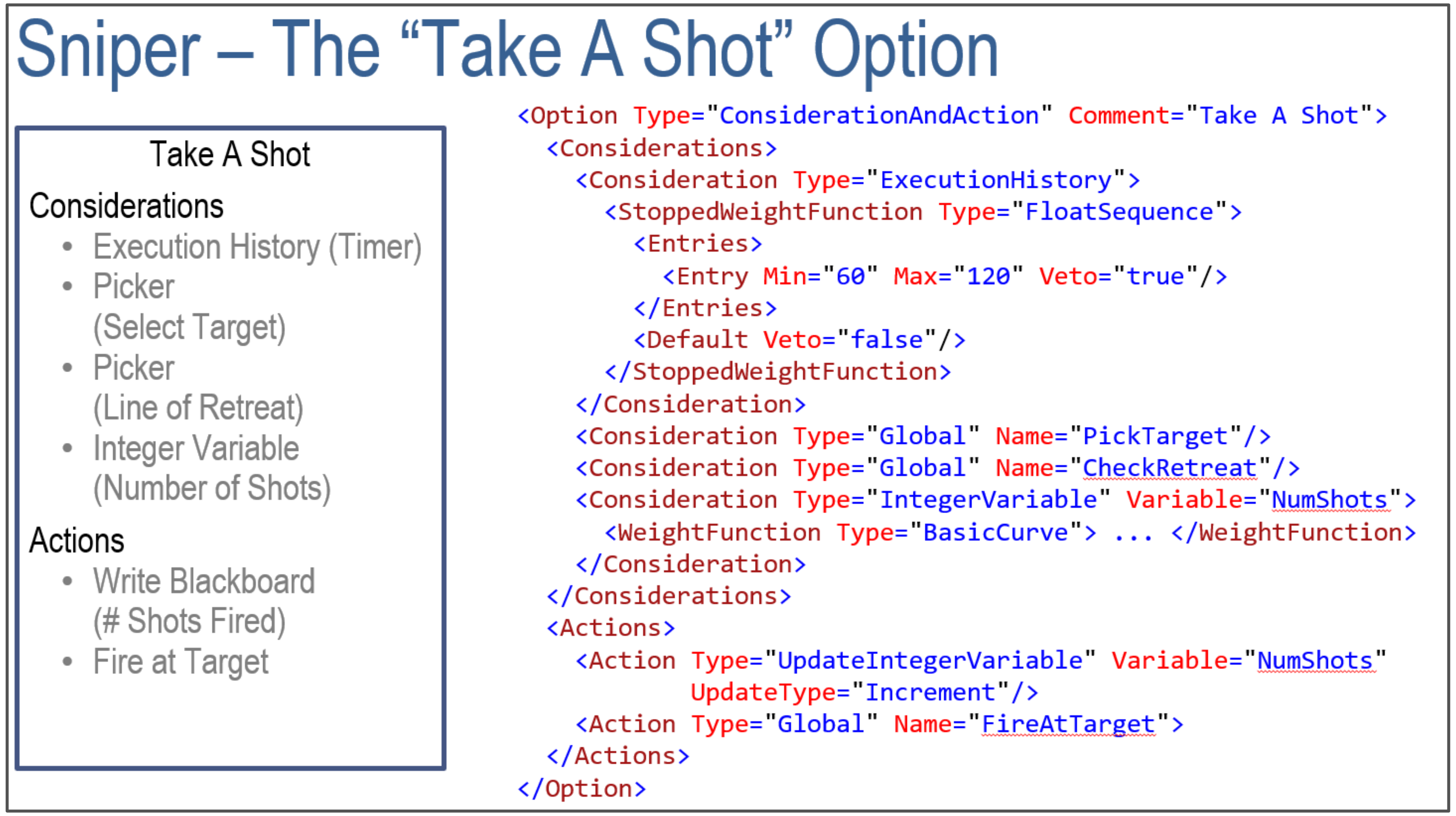
- This talk is about:
  - Why you should test
  - What I'm doing differently
  - Some tips and tricks that work for me
  - A few takeaways

Definitions

KYTHERA AI

GAIA: Game AI Architecture

- Modular AI architecture developed at Lockheed Martin
- Used for numerous different problems on numerous different simulations & game engines
- 2016 AI Summit talk
- Game AI Pro 3 article



Sniper – The "Take A Shot" Option

Take A Shot

Considerations
- Execution History (Timer)
- Picker (Select Target)
- Picker (Line of Retreat)
- Integer Variable (Number of Shots)

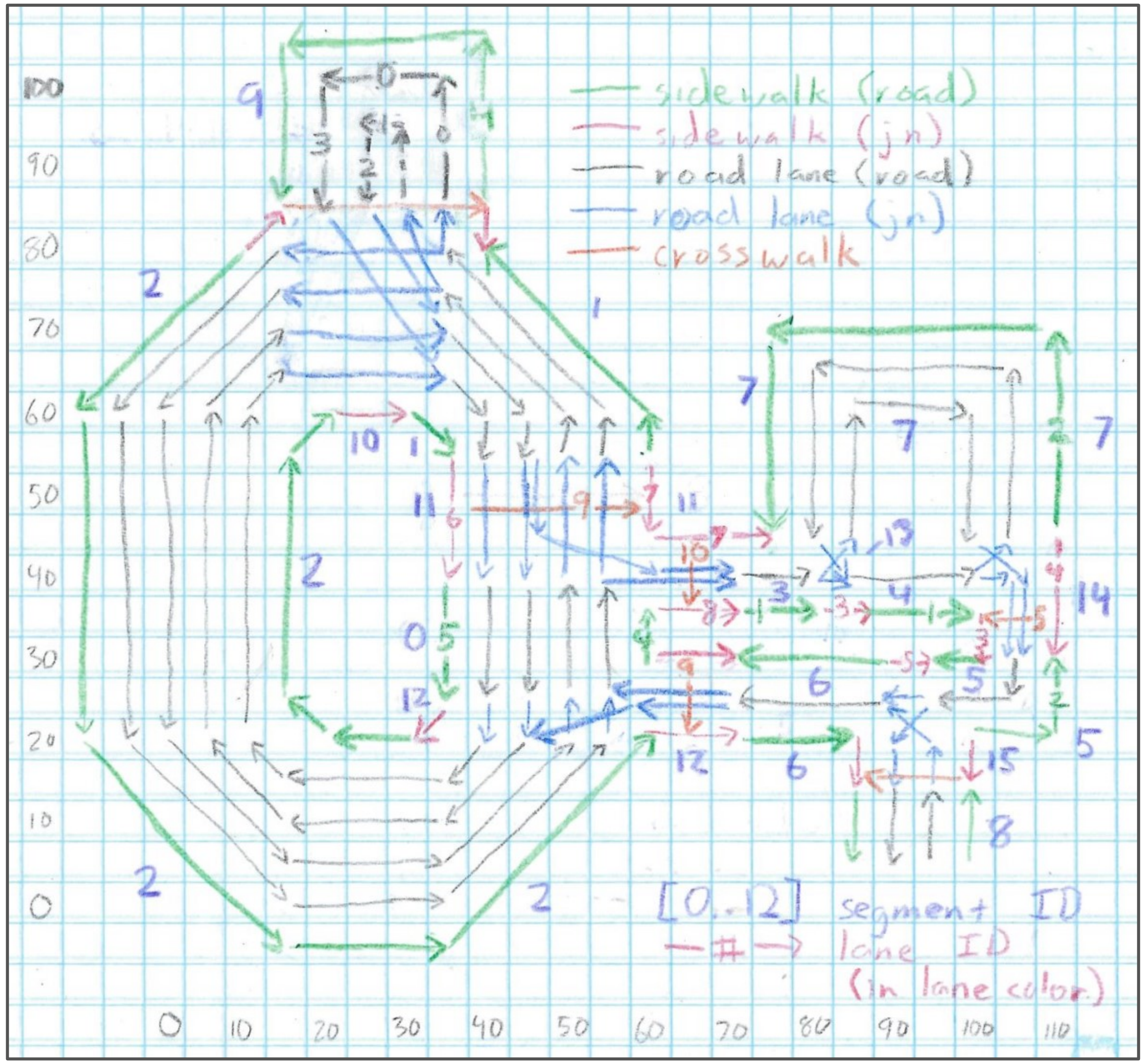
Actions
- Write Blackboard (# Shots Fired)
- Fire at Target

```
<Option Type="ConsiderationAndAction" Comment="Take A Shot">
  <Considerations>
    <Consideration Type="ExecutionHistory">
      <StoppedWeightFunction Type="FloatSequence">
        <Entries>
          <Entry Min="60" Max="120" Veto="true"/>
        </Entries>
        <Default Veto="false"/>
      </StoppedWeightFunction>
    </Consideration>
    <Consideration Type="Global" Name="PickTarget"/>
    <Consideration Type="Global" Name="CheckRetreat"/>
    <Consideration Type="IntegerVariable" Variable="NumShots">
      <WeightFunction Type="BasicCurve"> ... </WeightFunction>
    </Consideration>
  </Considerations>
  <Actions>
    <Action Type="UpdateIntegerVariable" Variable="NumShots"
            UpdateType="Increment"/>
    <Action Type="Global" Name="FireAtTarget">
  </Actions>
</Option>
```

Definitions

CSN: City Scale Navigation

- New feature under development at Kythera AI
- Fill large open worlds with ambient vehicles & pedestrians
- Graph-based spatial representation



Unit Tests (According to Conventional Wisdom)

A unit test is an ***automated*** piece of code that invokes a ***unit of work*** in the system and then checks a ***single assumption*** about the behavior of that unit of work.

A good unit test is:

- Fully automated
- Readable
- Maintainable
- Consistent
- Order-agnostic
- Fast
- Runs in memory
- Atomic

Your unit test library

- Is written alongside the code, in C++
- Covers as much code as possible
- Runs every time you build, every time you push, and again every night

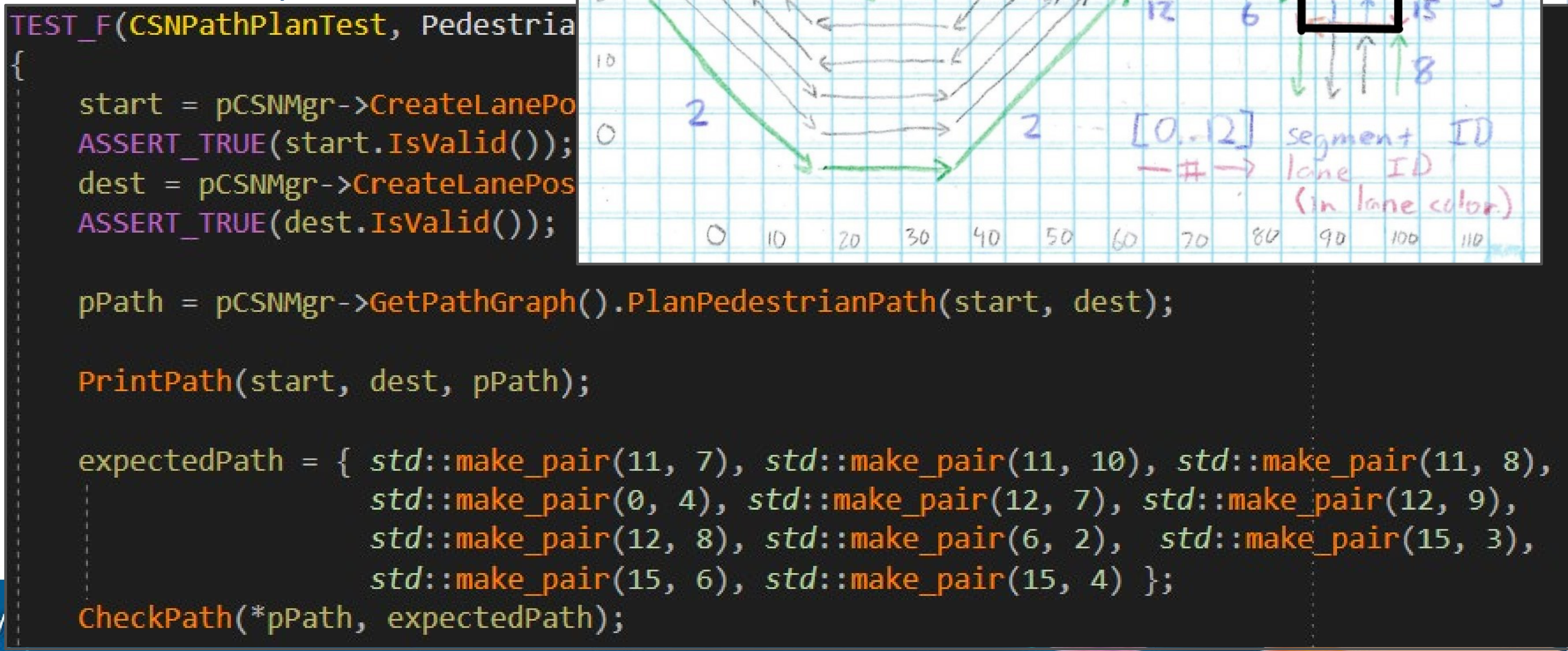
Source: www.artofunittesting.com/definition-of-a-unit-test

Why Test?

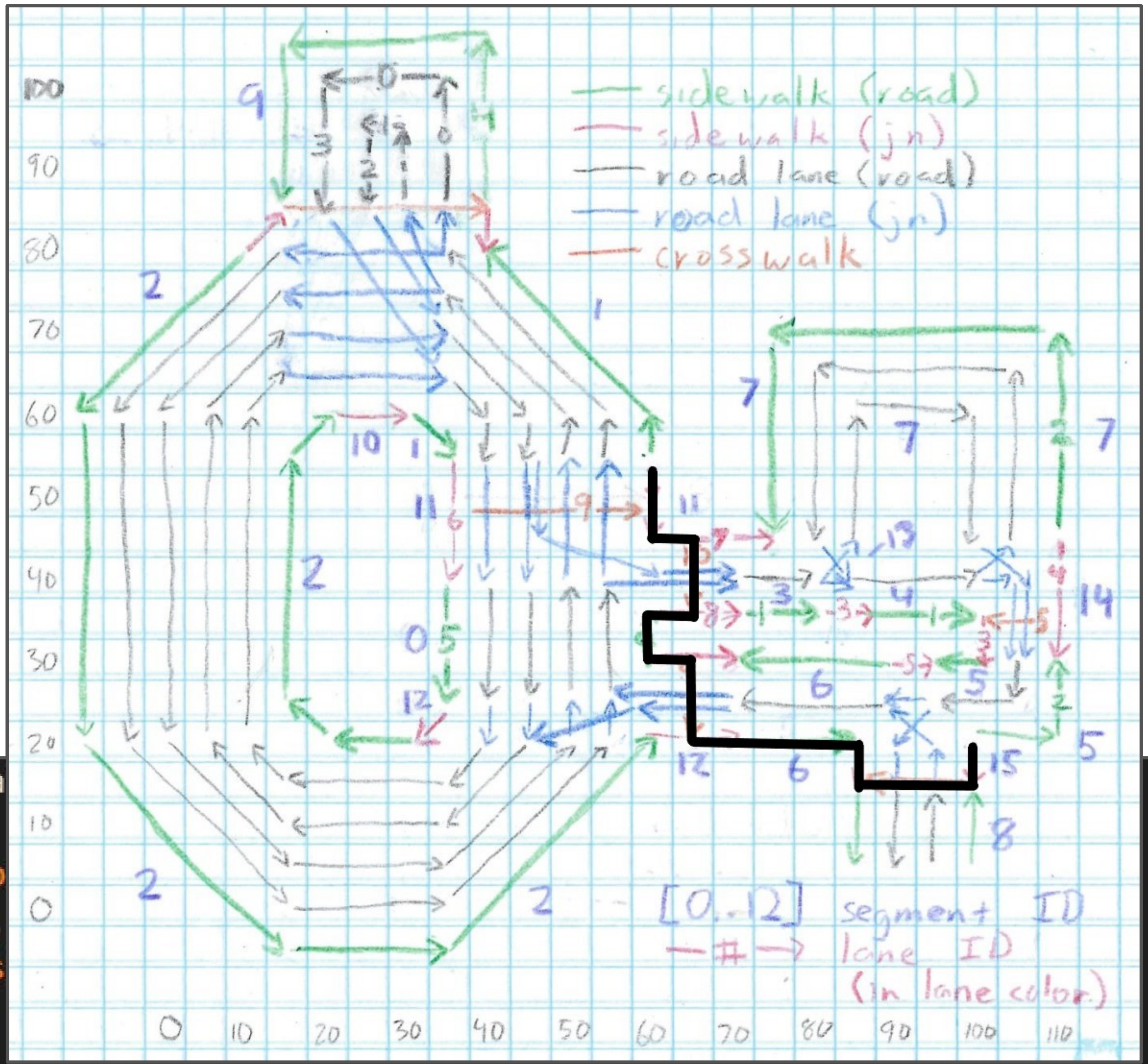
- Instant feedback
  - Catch mistakes right away
  - ***The best time to fix a bug is just after you wrote it***
  - ***That is also the <u>safest</u> time to fix it!***
- It's like the Easy button for bug finding!
  - Know exactly where the bug is
    - As opposed to getting it from QA, having to track it down
  - Instant repro case
    - As opposed to trying to make it happen in-game

Why Test?

- Document the code
  - Step in and see what's happening – fully instantiated!
  - Get up to speed when you come back to a feature
    - Or when you didn't write the feature!
- Example: CSN Path Planner
  - Works on CSN graph (rather than navmesh)
  - Intended for use in missions
  - Added to Kythera early
  - Wasn't hooked up in-game for almost a year…
  - I was on another project!!
  - "Does it handle crosswalks?"



```
TEST_F(CSNPathPlanTest, Pedestria
{
    start = pCSNMgr->CreateLanePo
    ASSERT_TRUE(start.IsValid());
    dest = pCSNMgr->CreateLanePos
    ASSERT_TRUE(dest.IsValid());

    pPath = pCSNMgr->GetPathGraph().PlanPedestrianPath(start, dest);

    PrintPath(start, dest, pPath);

    expectedPath = { std::make_pair(11, 7), std::make_pair(11, 10), std::make_pair(11, 8),
                     std::make_pair(0, 4), std::make_pair(12, 7), std::make_pair(12, 9),
                     std::make_pair(12, 8), std::make_pair(6, 2),  std::make_pair(15, 3),
                     std::make_pair(15, 6), std::make_pair(15, 4) };
    CheckPath(*pPath, expectedPath);
```



Why Test?

- Safety net when the code changes…
  - … and this is game development – the code ***always*** changes
- Example: Pedestrian Collision Avoidance
  - At very high LOD, pedestrians avoid one another
  - Requires a special avoidance component to be added/removed as LOD changes
  - Problem: When the player spawns in, the LOD changes from low to high…
    - First, spawn actors (creating an avoidance component)
    - Second, set the LOD on all actors (also creating an avoidance component)
    - Third, crash!
  - Reversing the order makes sense… but does it create another bug?
  - ***We have no tests for this… I don't know!!***

“Unit” Tests (According To Kevin)

KYTHERA AI

A “unit” test is an ***automated*** piece of code that invokes the system and tests it.

A good unit test is:

- Fully automated
- Readable
- Maintainable
- Consistent
- Order-agnostic

$\left.\right\}$ Critical!!

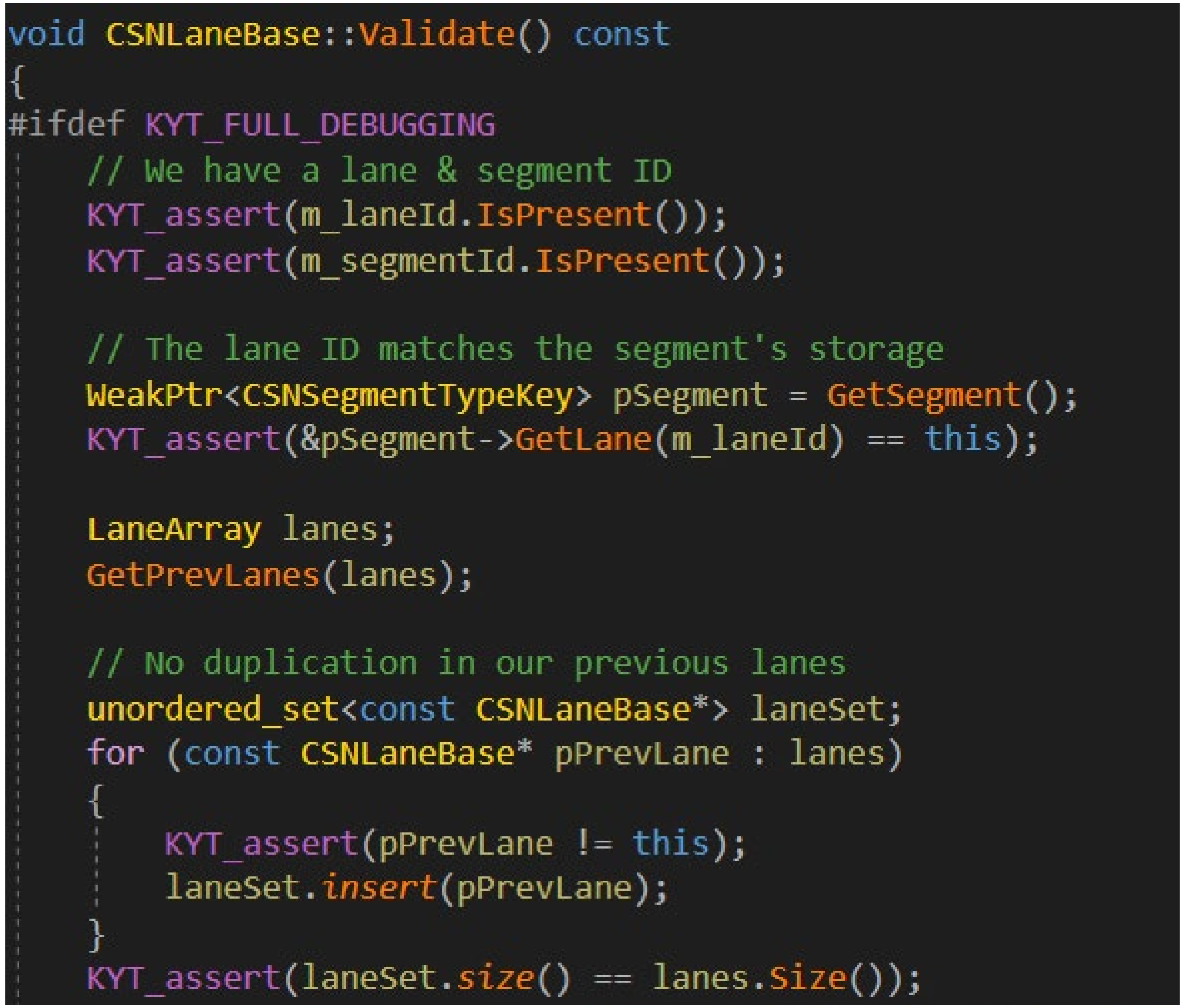
- Fast — Yes, to a point
- Runs in memory — Who cares?
- Atomic — This is actually bad!!

Tips & Tricks: Test Injection

KYTHERA AI

- The Problem:
  - Tests use public interfaces
  - Exposing things for the tests is a bad smell – you're testing the implementation details!
  - But, not everything can be tested via the API

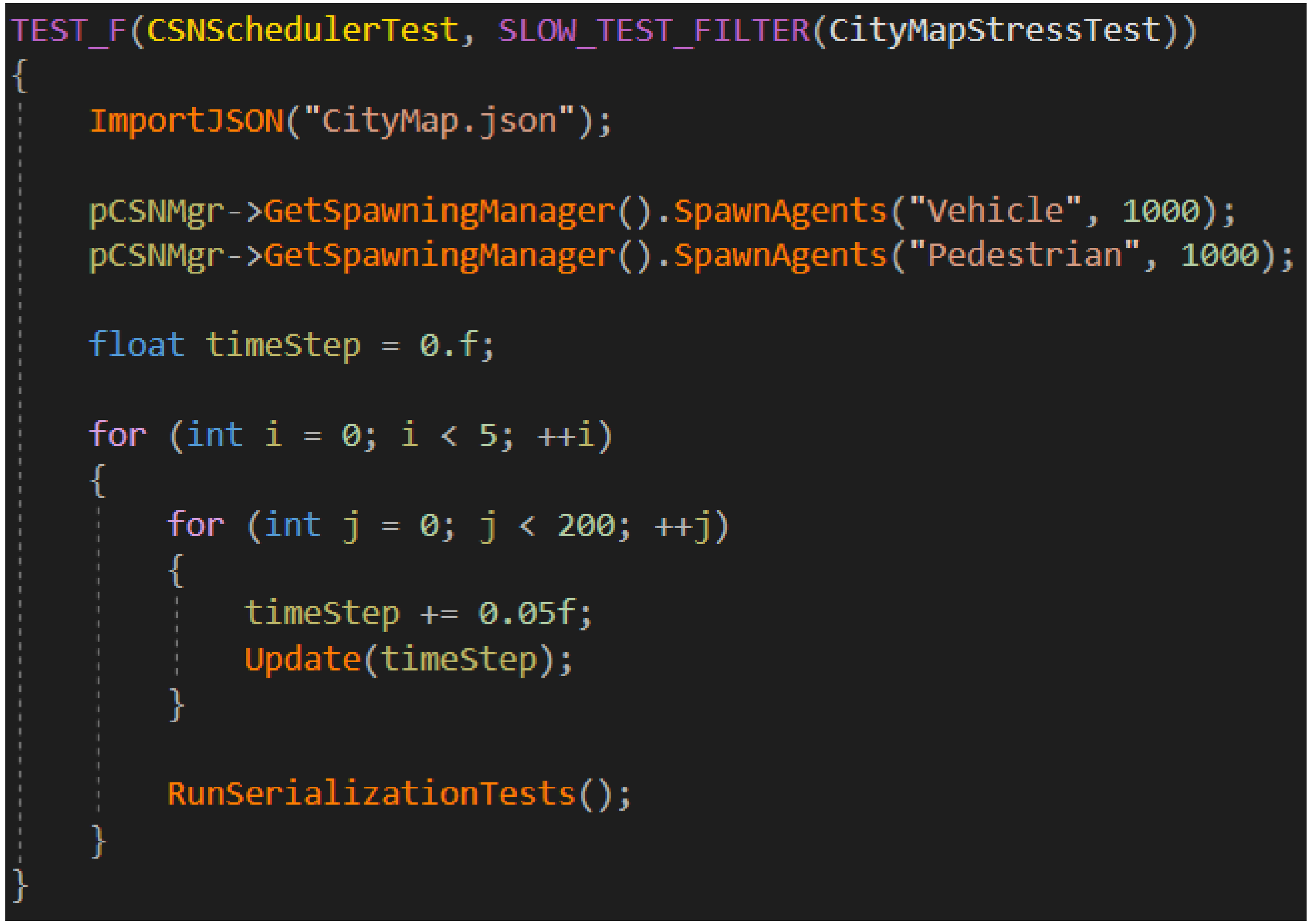
- Solution: ***Inject your tests into the code***
  - Asserts, Errors, Warnings make tests fail
    - (if unexpected)
  - Code checks that edge cases work correctly
  - Tests make the edge cases happen



```
void CSNLaneBase::Validate() const
{
#ifdef KYT_FULL_DEBUGGING
    // We have a lane & segment ID
    KYT_assert(m_laneId.IsPresent());
    KYT_assert(m_segmentId.IsPresent());

    // The lane ID matches the segment's storage
    WeakPtr<CSNSegmentTypeKey> pSegment = GetSegment();
    KYT_assert(&pSegment->GetLane(m_laneId) == this);

    LaneArray lanes;
    GetPrevLanes(lanes);

    // No duplication in our previous lanes
    unordered_set<const CSNLaneBase*> laneSet;
    for (const CSNLaneBase* pPrevLane : lanes)
    {
        KYT_assert(pPrevLane != this);
        laneSet.insert(pPrevLane);
    }
    KYT_assert(laneSet.size() == lanes.Size());
```

More Tips & Tricks

- Stress Tests:
  - Load a large map, create lots of entities, run lots of updates
  - See if anything breaks (Test Injection)
  - Vary the frame rate
- Slow Tests:
  - Compiler directive to enable/disable
  - Enabled in the nightly build & CI
  - Can be enabled by a developer if needed



```
TEST_F(CSNSchedulerTest, SLOW_TEST_FILTER(CityMapStressTest))
{
    ImportJSON("CityMap.json");

    pCSNMgr->GetSpawningManager().SpawnAgents("Vehicle", 1000);
    pCSNMgr->GetSpawningManager().SpawnAgents("Pedestrian", 1000);

    float timeStep = 0.f;

    for (int i = 0; i < 5; ++i)
    {
        for (int j = 0; j < 200; ++j)
        {
            timeStep += 0.05f;
            Update(timeStep);
        }

        RunSerializationTests();
    }
}
```

Why We Fail / Takeaways

- Too Big / Too Hard / Don't Know Where to Begin
  - Don't angst about perfect coverage
  - Start with your next line of code
  - Better yet, your next bug
- Test Maintenance > Test Payoff
  - Think about your level of granularity
  - Think about building validation into the code
- Lack of Buy-In / Lack of Discipline
  - There are times… but you have to circle back
  - ***We don't have time to <u>not</u> write tests!***

KYTHERA AI

Contact us

t:	+44(0)131 2902621
e:	contact@kythera.ai

Follow us

@KytheraAI

/company/kythera-ai

Visit us

www.kythera.ai/gdc21

Kevin Dill: kevind@kythera.ai

Click to Add Title

- Click to add text
  - Second level
    - Third level
      - Fourth level
        - Fifth level

Unit Tests (According to Conventional Wisdom)

A unit test is an ***automated*** piece of code that invokes a ***unit of work*** in the system and then checks a ***single assumption*** about the behavior of that unit of work.

A good unit test is:

- Fully automated
- Readable
- Maintainable
- Consistent
- Order-agnostic
- Fast
- Runs in memory
- Atomic

Your unit test library

- Is written alongside the code, in C++
- Covers as much code as possible
- Runs every time you build, every time you push, and again every night

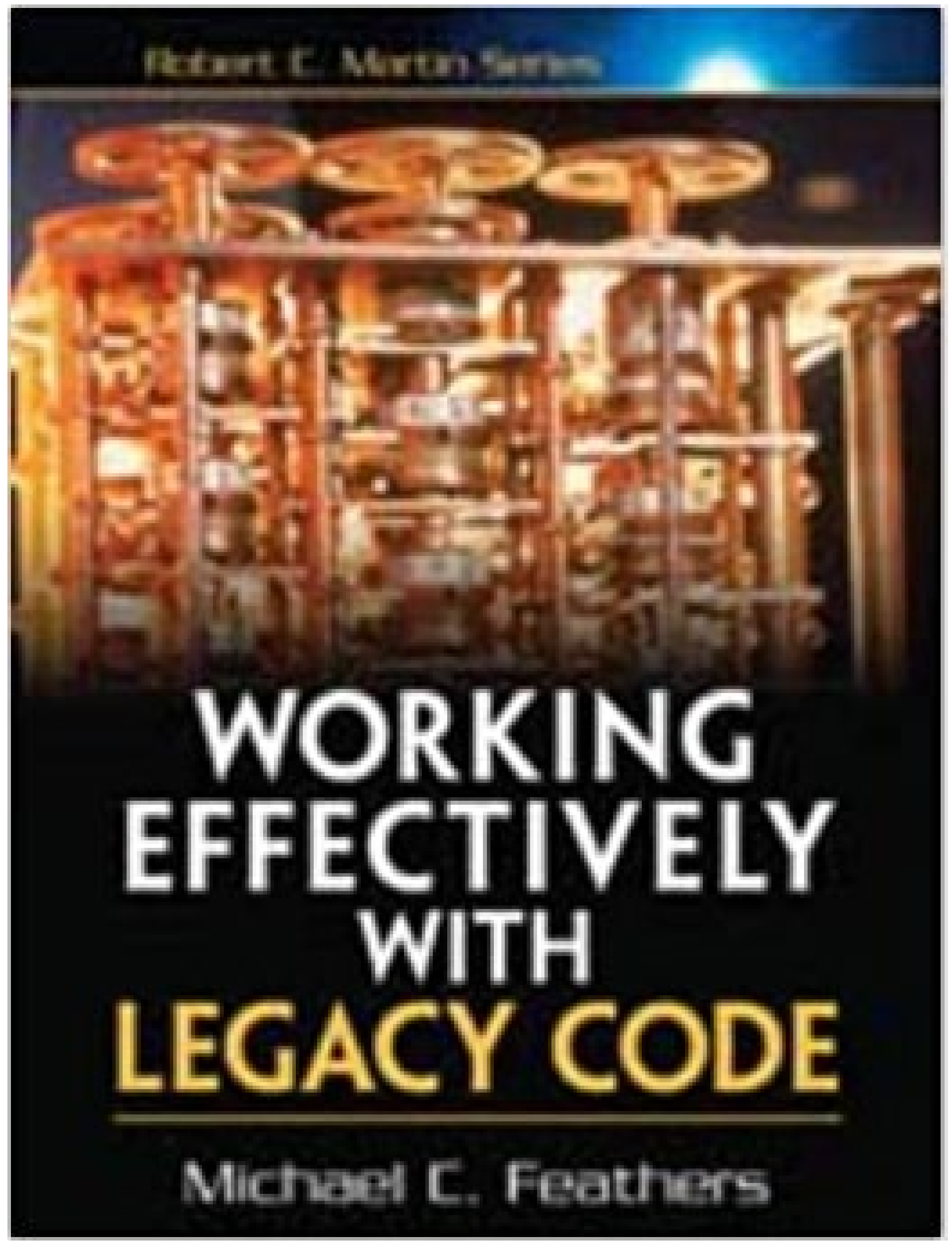
Other Kinds of Tests:

- Integration
- Functional
- End-To-End

Source: www.artofunittesting.com/definition-of-a-unit-test

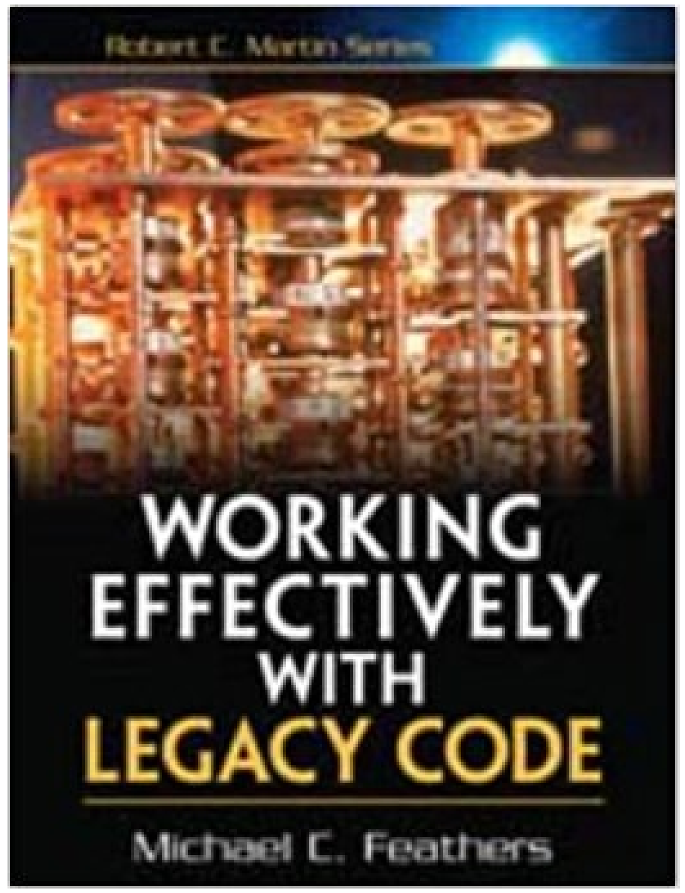
Notes & Disclaimers

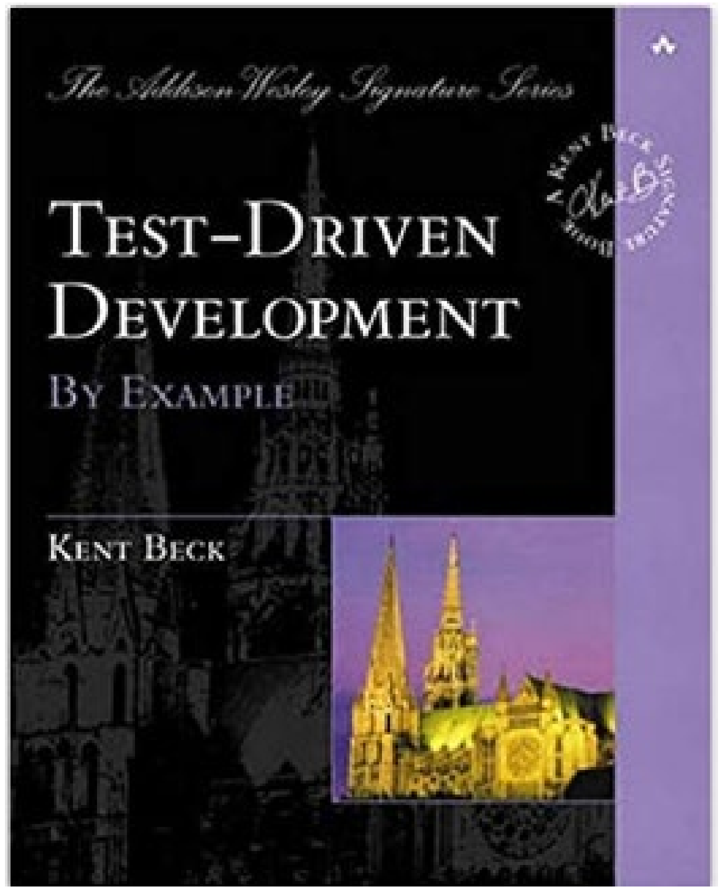
- I am not "Uncle Bob" Martin or Michael Feathers
- What I am going to give you is not the stock software engineering pitch…
  - What has worked for me
  - Actual payoff I've experienced
  - I do give you one slide that's by the book ;)
- Some of it will fly in the face of conventional wisdom
  - As with anything, YMMV

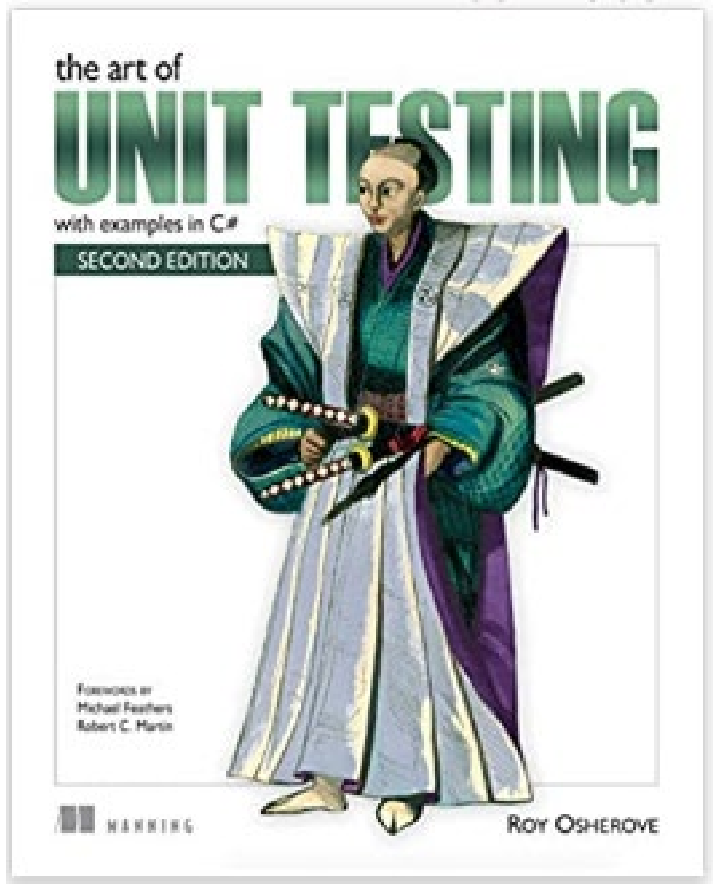


References

- GDC 2006: Backwards is Forward: Making Better Games with Test-Driven Development (Sean Houghton & Noel Llopis)
- GDC 2014: Practical Unit Tests (Andrew Fray)
- GDC 2021: Automated Testing Roundtable







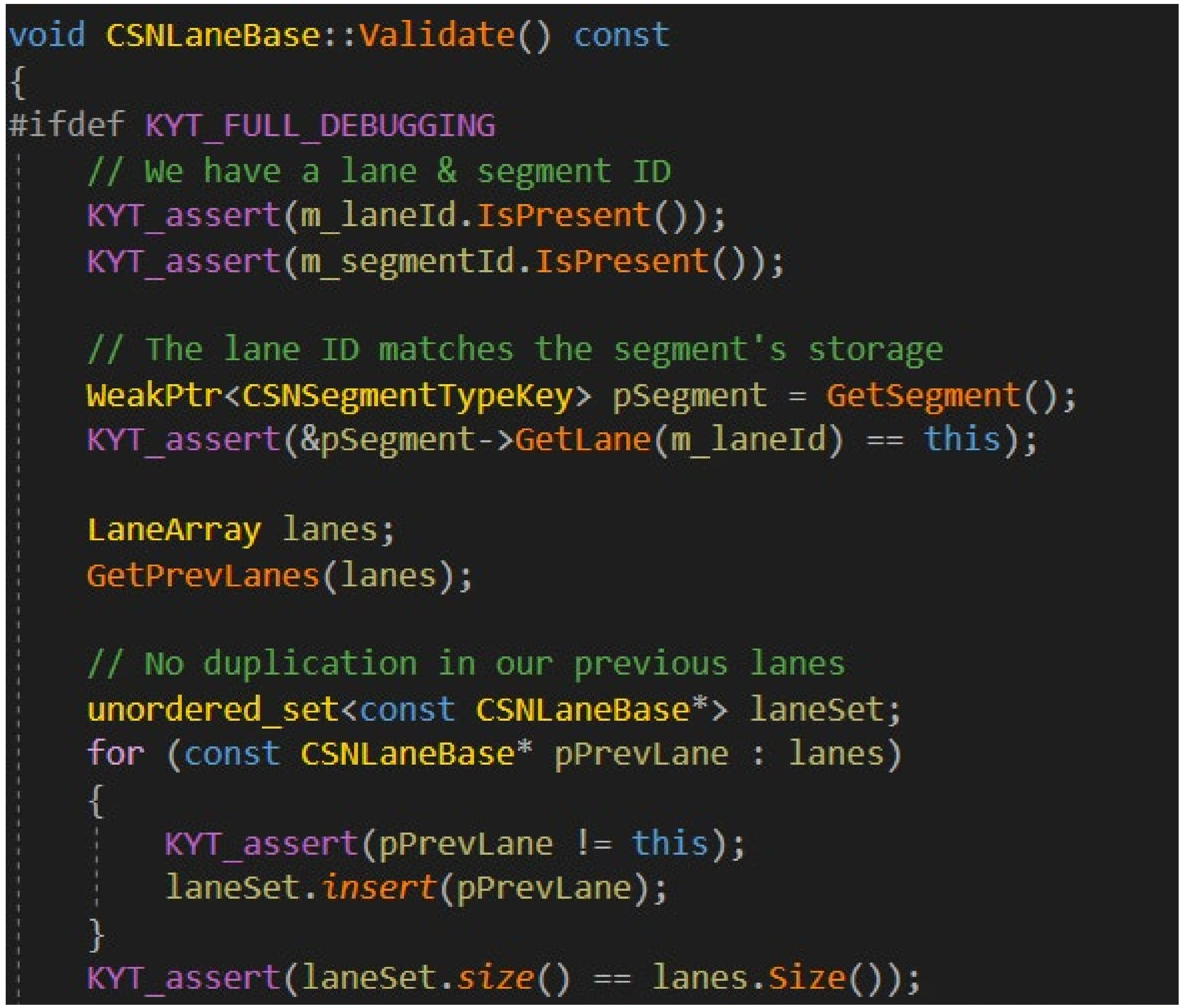
Why Bother

KYTHERA AI

- Change in mindset
  - Code is not complete until it has been tested
  - Loading a level and setting a few breakpoints is not good enough
  - ***Testing your code is not QA's job – it's yours!!***

Test Injection

- The Problem:
  - Tests use public interface
  - May be restricted to API
- Solutions:
  - Asserts
  - Errors & Warnings
    - Unexpected: trigger failure
    - Expected: confirm they occur
  - `#ifdef` validation blocks
  - Memory leaks



```
void CSNLaneBase::Validate() const
{
#ifdef KYT_FULL_DEBUGGING
    // We have a lane & segment ID
    KYT_assert(m_laneId.IsPresent());
    KYT_assert(m_segmentId.IsPresent());

    // The lane ID matches the segment's storage
    WeakPtr<CSNSegmentTypeKey> pSegment = GetSegment();
    KYT_assert(&pSegment->GetLane(m_laneId) == this);

    LaneArray lanes;
    GetPrevLanes(lanes);

    // No duplication in our previous lanes
    unordered_set<const CSNLaneBase*> laneSet;
    for (const CSNLaneBase* pPrevLane : lanes)
    {
        KYT_assert(pPrevLane != this);
        laneSet.insert(pPrevLane);
    }
    KYT_assert(laneSet.size() == lanes.Size());
```

Time and Time Again…

KYTHERA AI

- "Unit testing doesn't really work in games"
- "Some systems just aren't easy to test"
- "It's hard to write tests in C++"
- "Games are too complex" / "Games change too much"
- "We don't have time to write tests"

Excuses!!!

- It can be done – I'm doing it
- It will improve your velocity – even if done imperfectly
- ***You don't have time to not write tests***

Still More Tricks

KYTHERA AI

- Performance Validation:
  - Set up a dedicated machine
  - For each test, track: elapsed time, memory usage, number of instructions, etc.
  - Send an alert if those numbers increase too much

Why Bother

KYTHERA AI

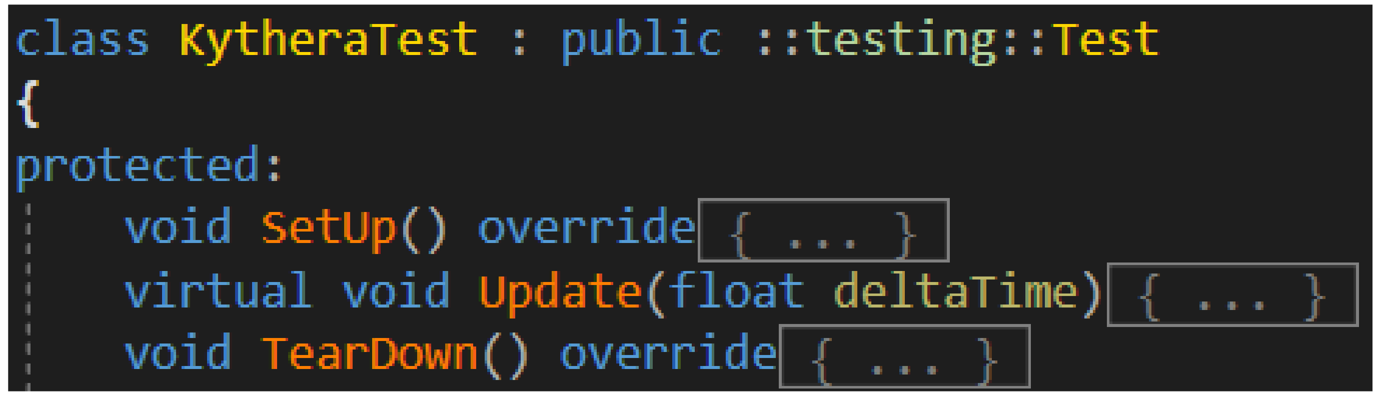
- Better code!
  - Get your interfaces right
  - Decoupled, well encapsulated, modular
  - Maintain discipline

Still More Tricks

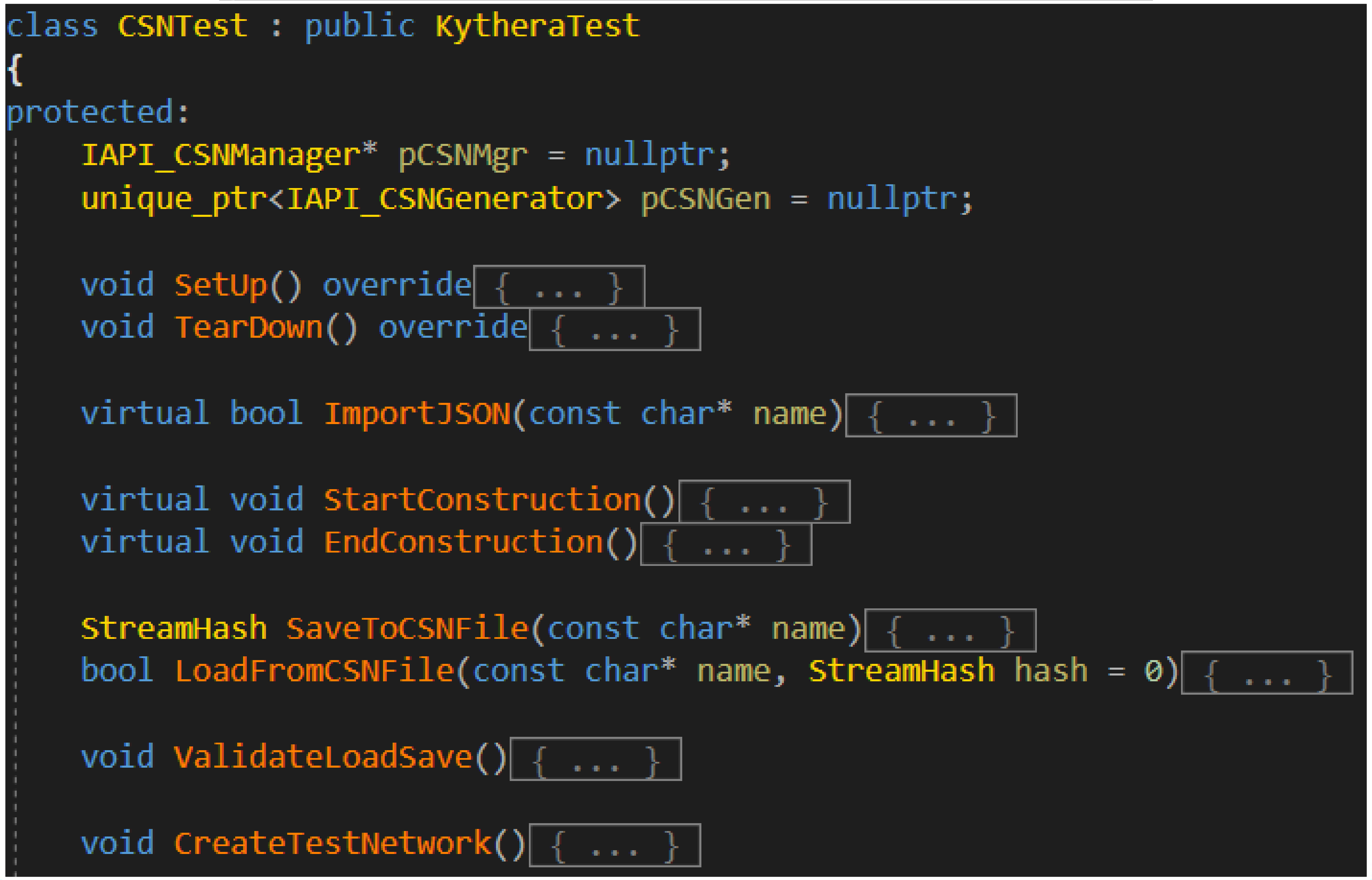
- Memory Leaks:
  - Override new/delete
  - Count allocations / deallocations (by object size)
  - On exit, make sure that they match!
- Performance Validation (plays nice with Stress Tests):
  - Set up a dedicated machine
  - For each test, track: elapsed time, memory usage, number of instructions, etc.
  - Send an alert if those numbers increase too much

Fixtures (Kythera / CSN)

- Automatically created/destroyed for each test
- Contains:
  - SetUp() / TearDown()
  - Frequently used data
  - Helper functions



```
class KytheraTest : public ::testing::Test
{
protected:
    void SetUp() override { ... }
    virtual void Update(float deltaTime) { ... }
    void TearDown() override { ... }
```

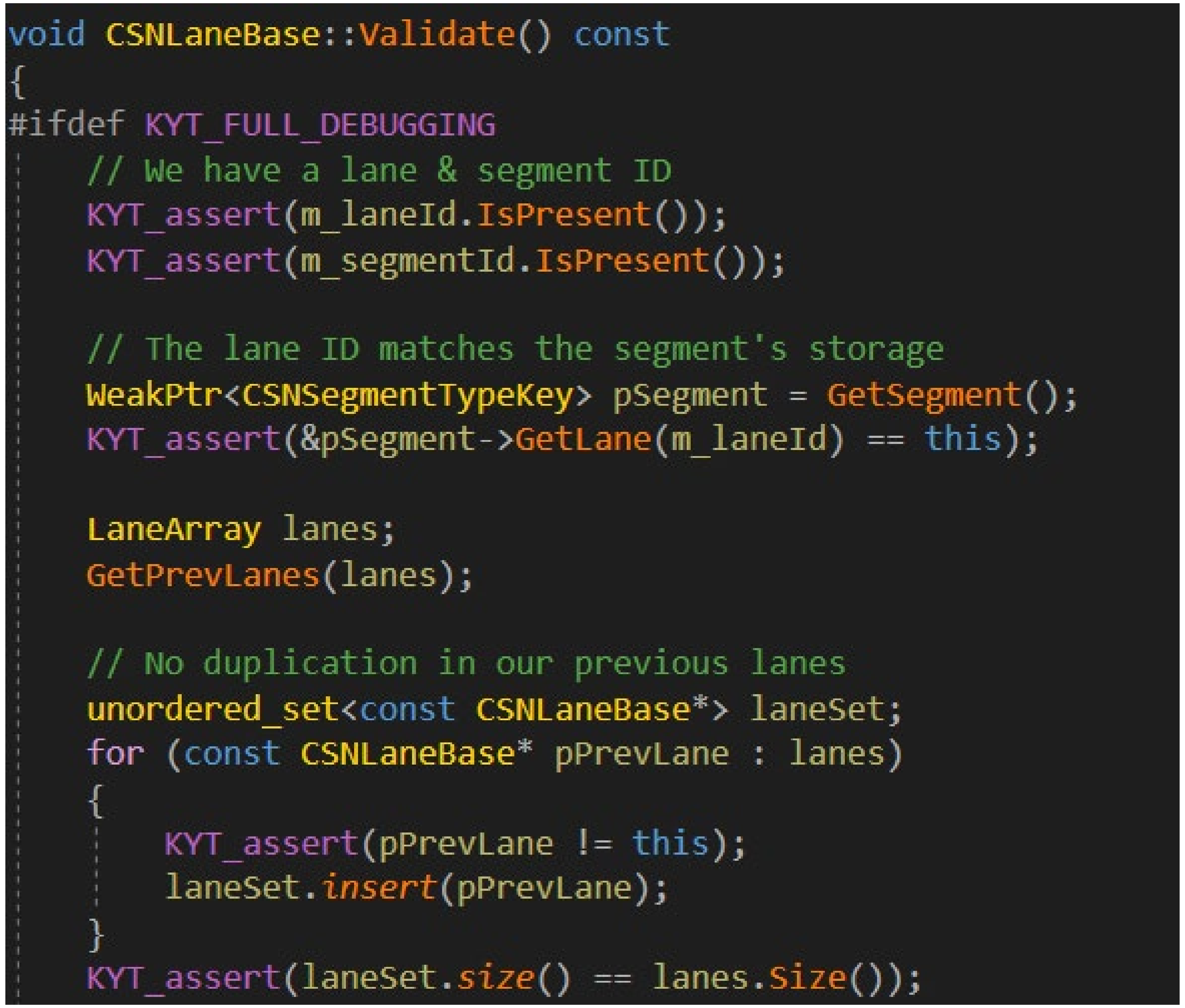


```
class CSNTest : public KytheraTest
{
protected:
    IAPI_CSNManager* pCSNMgr = nullptr;
    unique_ptr<IAPI_CSNGenerator> pCSNGen = nullptr;

    void SetUp() override { ... }
    void TearDown() override { ... }

    virtual bool ImportJSON(const char* name) { ... }

    virtual void StartConstruction() { ... }
    virtual void EndConstruction() { ... }

    StreamHash SaveToCSNFile(const char* name) { ... }
    bool LoadFromCSNFile(const char* name, StreamHash hash = 0) { ... }

    void ValidateLoadSave() { ... }

    void CreateTestNetwork() { ... }
```

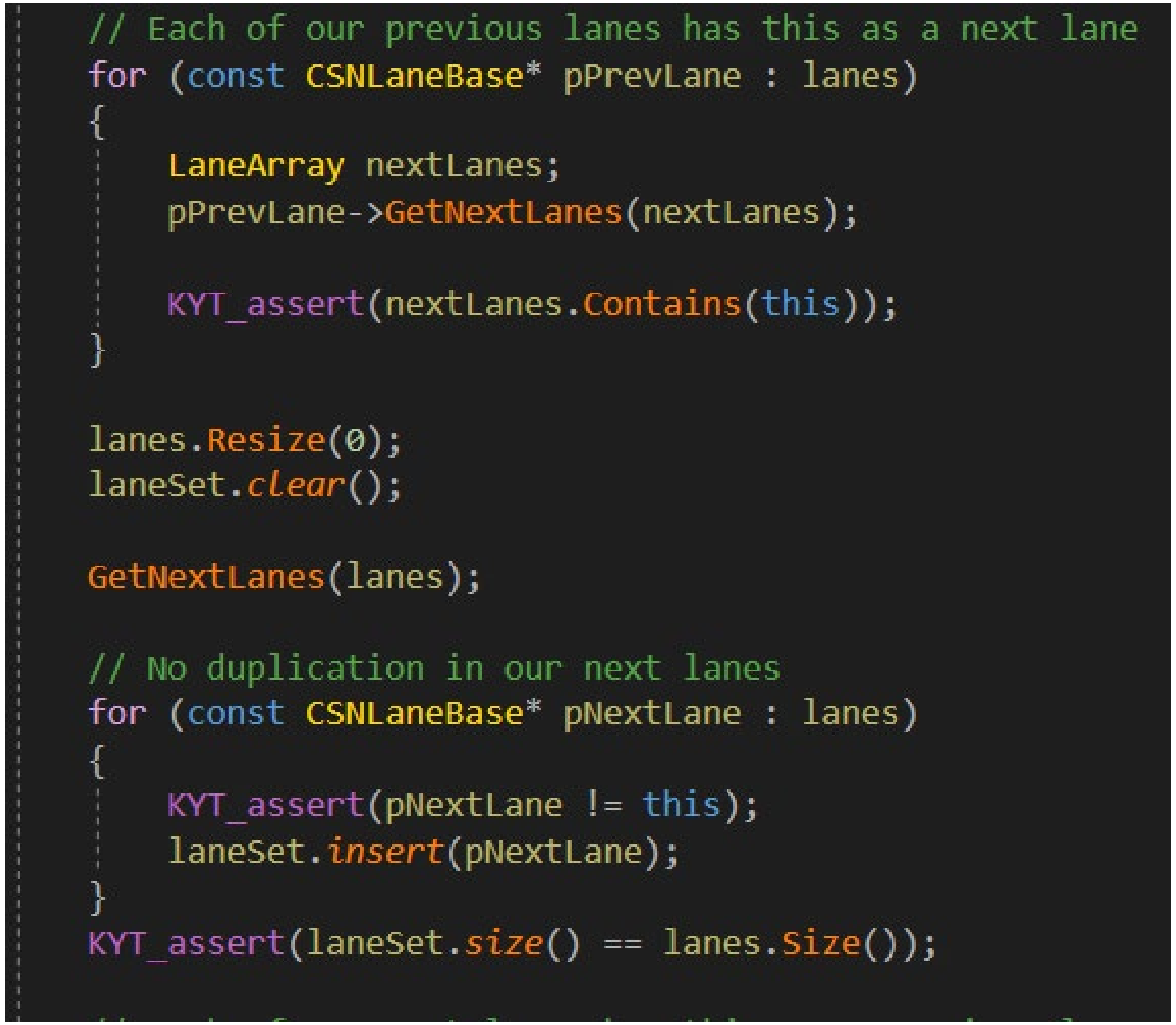
Payoff

- Safety net when code is extended or modified
  - This is game development
  - The code always changes
- Example: CSN Level Of Detail (LOD)
  - New feature: Pedestrian collision avoidance
    - Only at high LOD
    - Requires a special avoidance component
  - Problem: When LOD changes from very low to very high…
    - First, spawn actors (creating an avoidance component)
    - Second, set the LOD on all actors (also creating an avoidance component
    - Third, crash!
  - Reversing the order makes sense… but is it safe?

Leverage Asserts (CSN)

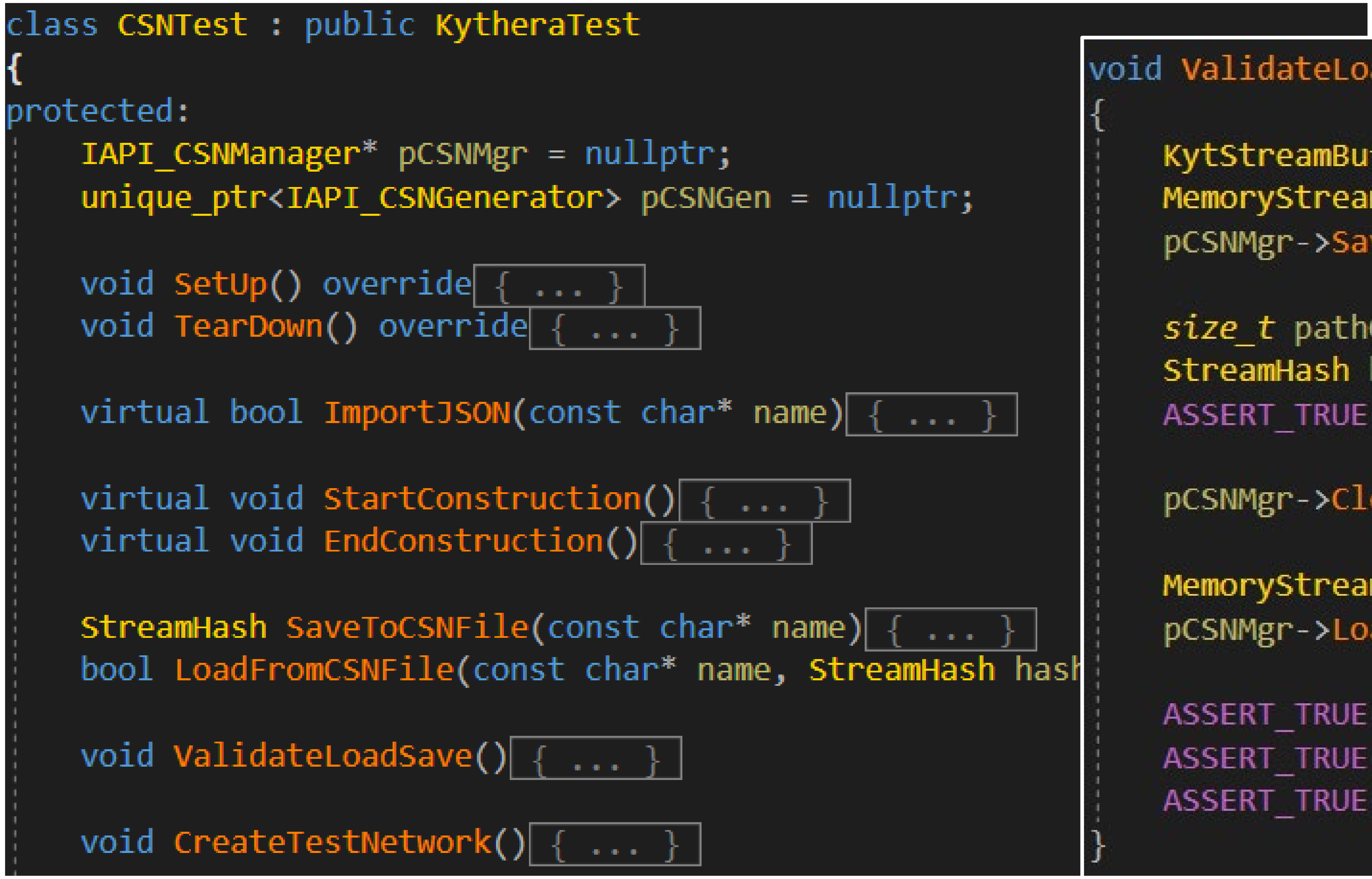


```
void CSNLaneBase::Validate() const
{
#ifdef KYT_FULL_DEBUGGING
    // We have a lane & segment ID
    KYT_assert(m_laneId.IsPresent());
    KYT_assert(m_segmentId.IsPresent());

    // The lane ID matches the segment's storage
    WeakPtr<CSNSegmentTypeKey> pSegment = GetSegment();
    KYT_assert(&pSegment->GetLane(m_laneId) == this);

    LaneArray lanes;
    GetPrevLanes(lanes);

    // No duplication in our previous lanes
    unordered_set<const CSNLaneBase*> laneSet;
    for (const CSNLaneBase* pPrevLane : lanes)
    {
        KYT_assert(pPrevLane != this);
        laneSet.insert(pPrevLane);
    }
    KYT_assert(laneSet.size() == lanes.Size());
```

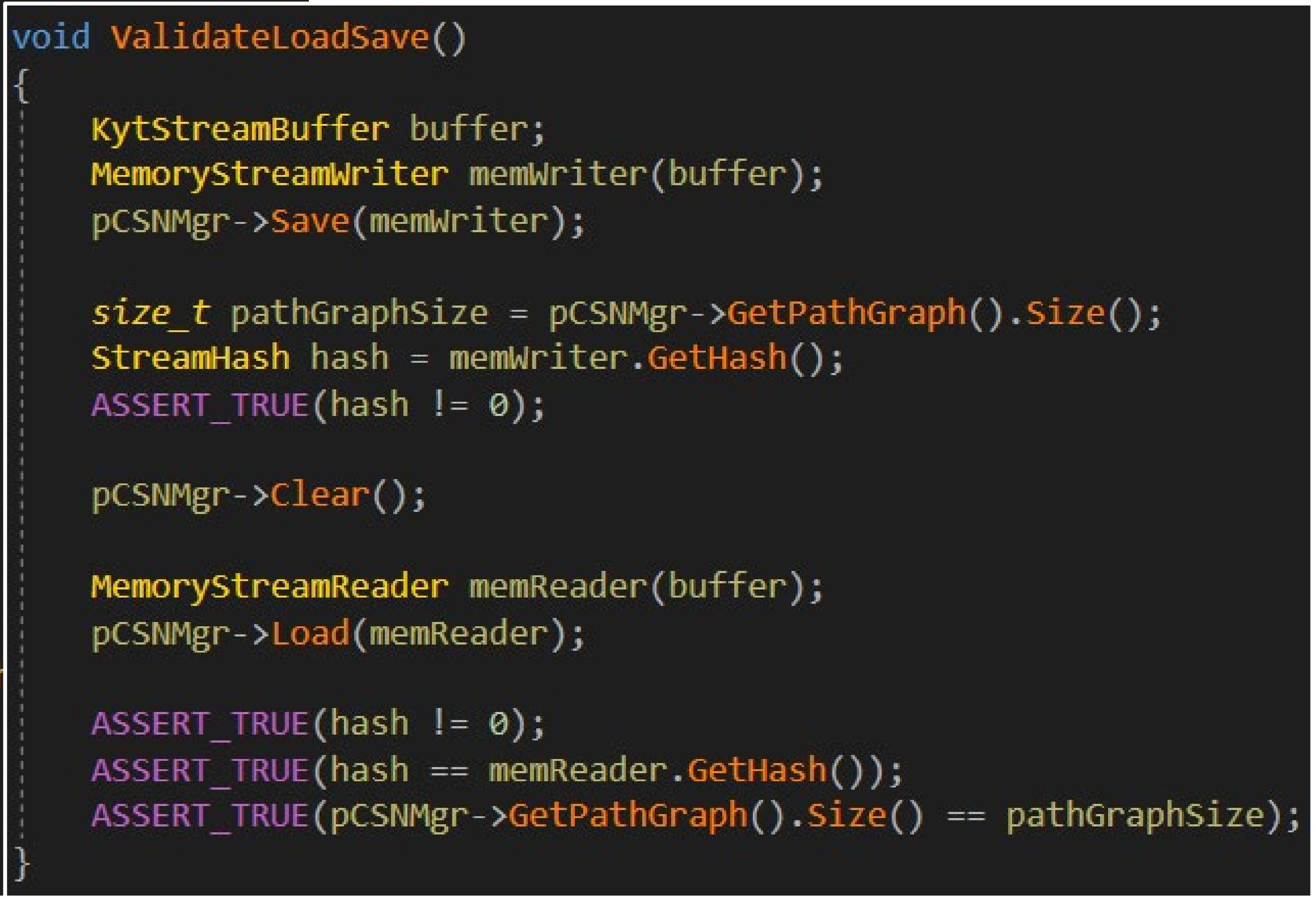


```
    // Each of our previous lanes has this as a next lane
    for (const CSNLaneBase* pPrevLane : lanes)
    {
        LaneArray nextLanes;
        pPrevLane->GetNextLanes(nextLanes);

        KYT_assert(nextLanes.Contains(this));
    }

    lanes.Resize(0);
    laneSet.clear();

    GetNextLanes(lanes);

    // No duplication in our next lanes
    for (const CSNLaneBase* pNextLane : lanes)
    {
        KYT_assert(pNextLane != this);
        laneSet.insert(pNextLane);
    }
    KYT_assert(laneSet.size() == lanes.Size());
```

Fixtures (Kythera / CSN)

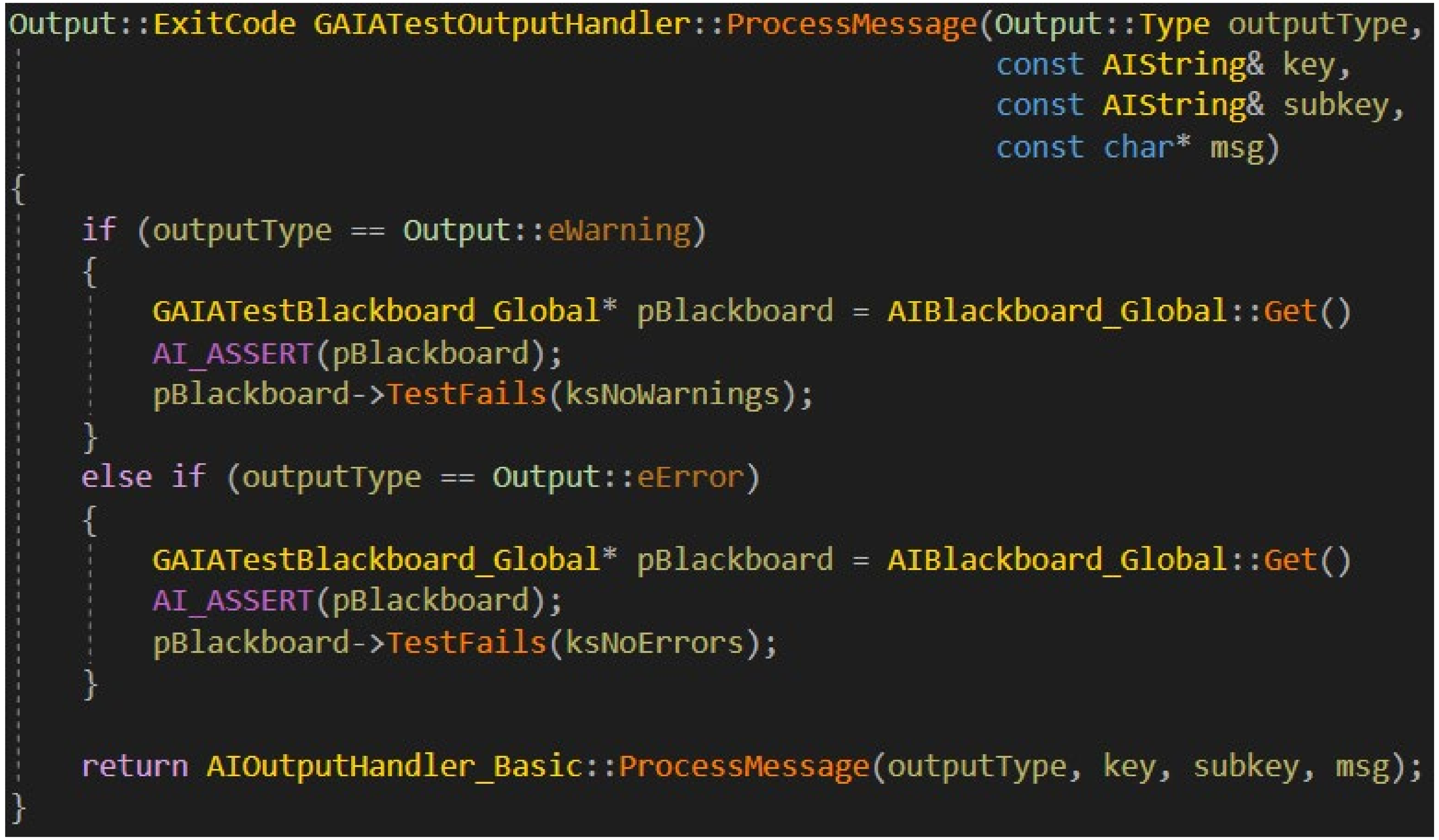


```
class CSNTest : public KytheraTest
{
protected:
    IAPI_CSNManager* pCSNMgr = nullptr;
    unique_ptr<IAPI_CSNGenerator> pCSNGen = nullptr;

    void SetUp() override { ... }
    void TearDown() override { ... }

    virtual bool ImportJSON(const char* name) { ... }

    virtual void StartConstruction() { ... }
    virtual void EndConstruction() { ... }

    StreamHash SaveToCSNFile(const char* name) { ... }
    bool LoadFromCSNFile(const char* name, StreamHash hash

    void ValidateLoadSave() { ... }

    void CreateTestNetwork() { ... }
```



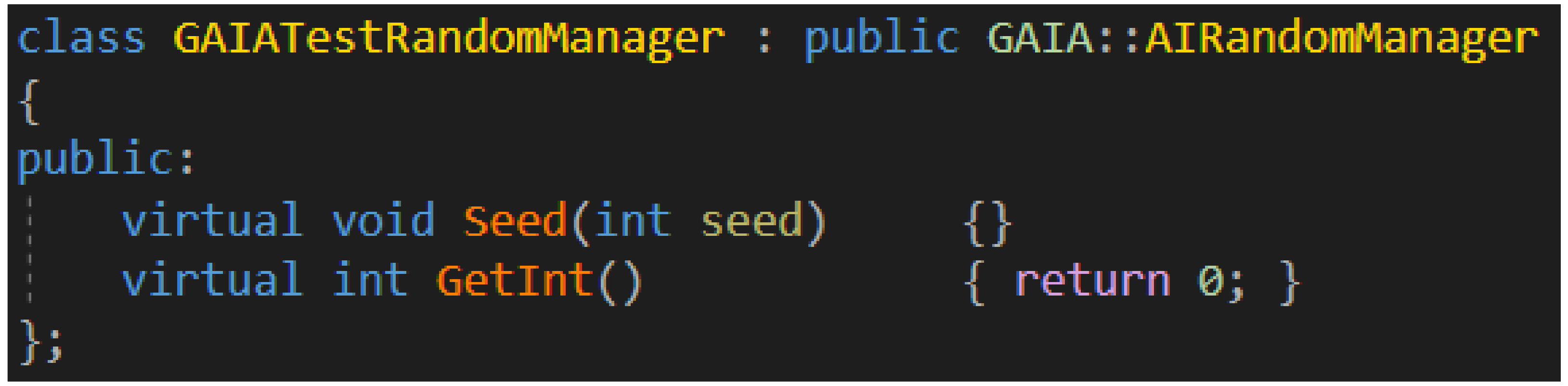
```
void ValidateLoadSave()
{
    KytStreamBuffer buffer;
    MemoryStreamWriter memWriter(buffer);
    pCSNMgr->Save(memWriter);

    size_t pathGraphSize = pCSNMgr->GetPathGraph().Size();
    StreamHash hash = memWriter.GetHash();
    ASSERT_TRUE(hash != 0);

    pCSNMgr->Clear();

    MemoryStreamReader memReader(buffer);
    pCSNMgr->Load(memReader);

    ASSERT_TRUE(hash != 0);
    ASSERT_TRUE(hash == memReader.GetHash());
    ASSERT_TRUE(pCSNMgr->GetPathGraph().Size() == pathGraphSize);
}
```

Errors & Warnings (GAIA)



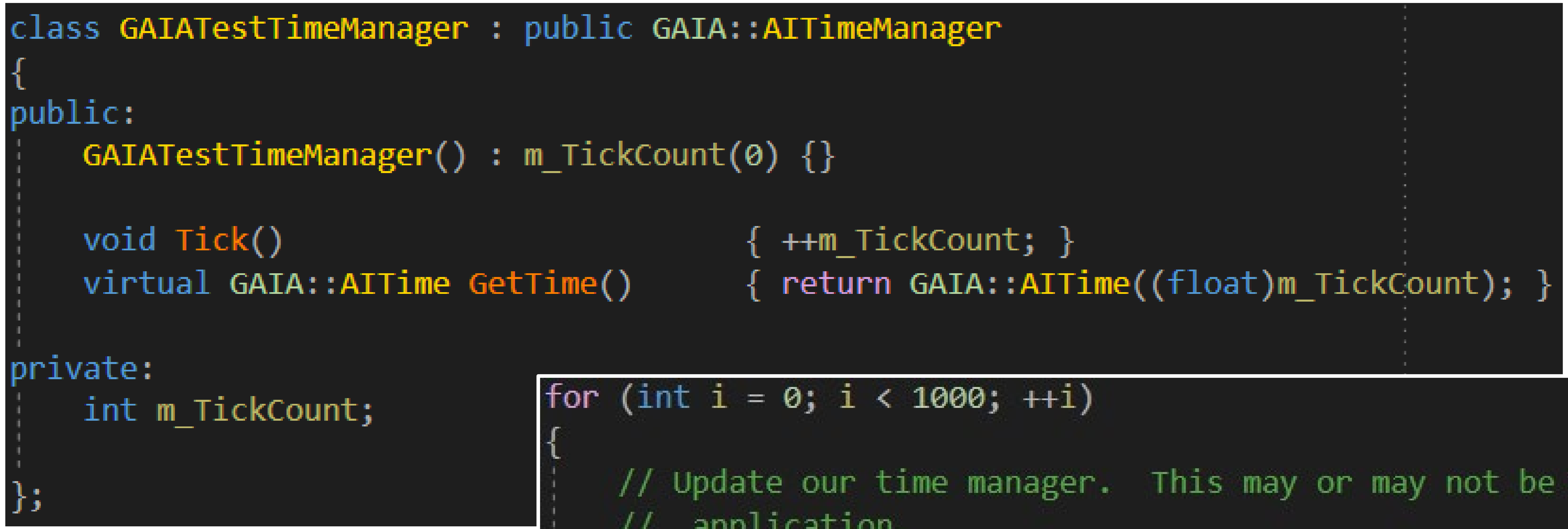
```
Output::ExitCode GAIATestOutputHandler::ProcessMessage(Output::Type outputType,
                                                       const AIString& key,
                                                       const AIString& subkey,
                                                       const char* msg)
{
    if (outputType == Output::eWarning)
    {
        GAIATestBlackboard_Global* pBlackboard = AIBlackboard_Global::Get()
        AI_ASSERT(pBlackboard);
        pBlackboard->TestFails(ksNoWarnings);
    }
    else if (outputType == Output::eError)
    {
        GAIATestBlackboard_Global* pBlackboard = AIBlackboard_Global::Get()
        AI_ASSERT(pBlackboard);
        pBlackboard->TestFails(ksNoErrors);
    }

    return AIOutputHandler_Basic::ProcessMessage(outputType, key, subkey, msg);
}
```

Mocks (GAIA)

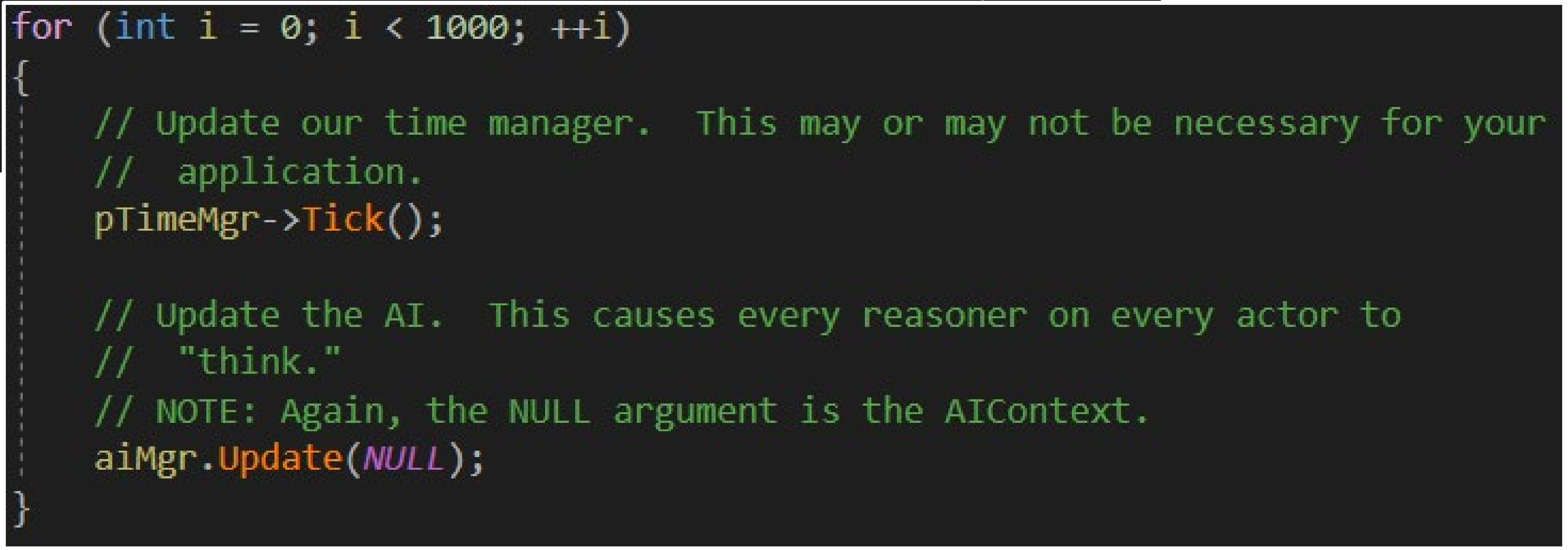


```
class GAIATestRandomManager : public GAIA::AIRandomManager
{
public:
    virtual void Seed(int seed)     {}
    virtual int GetInt()            { return 0; }
};
```

Mocks (GAIA)



```
class GAIATestTimeManager : public GAIA::AITimeManager
{
public:
    GAIATestTimeManager() : m_TickCount(0) {}

    void Tick()                           { ++m_TickCount; }
    virtual GAIA::AITime GetTime()        { return GAIA::AITime((float)m_TickCount); }

private:
    int m_TickCount;

};
```



```
for (int i = 0; i < 1000; ++i)
{
    // Update our time manager.  This may or may not be necessary for your
    //  application.
    pTimeMgr->Tick();

    // Update the AI.  This causes every reasoner on every actor to
    //  "think."
    // NOTE: Again, the NULL argument is the AIContext.
    aiMgr.Update(NULL);
}
```