

---

the  
gamedesigninitiative  
at cornell university

---

# Debugging Strategies

# There are Two Main Strategies

---

- **Confirmation**

- Confirm everything you believe to be true
- Find the thing that is not actually true
- In worse case, have to look at every line of code

- **Binary Search**

- Identify where the code is working properly
- Identify where the code is not working properly
- Limit confirmation to the space in between

# There are Two Main Strategies

---

- **Confirmation**

- Confirm everything you believe to be true
- Find the thing that is not actually true
- In worse case, have to look at

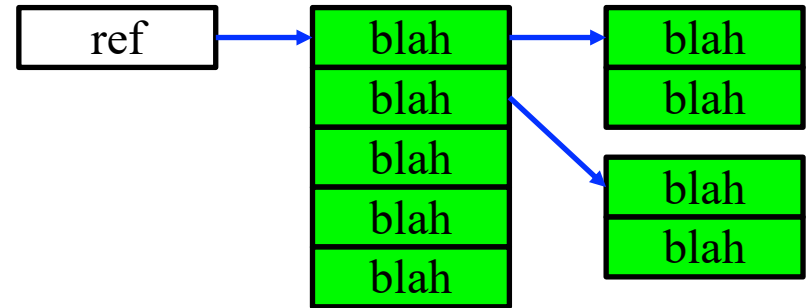
● Everything else is a fancy tool to do this

- Identify where the code is working properly
- Identify where the code is not working properly
- Limit confirmation to the space in between

# The Challenge of Finding Errors

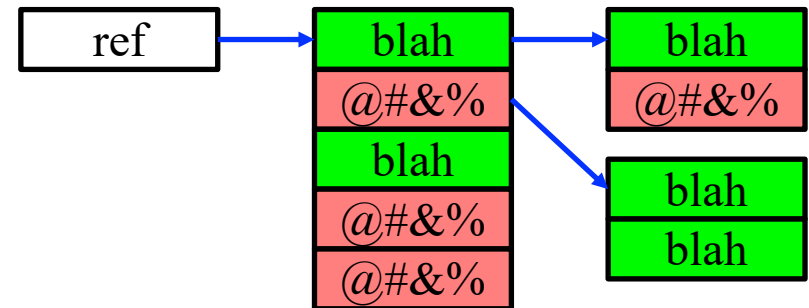
---

- *Access errors* are the hardest
  - Refer to object in memory
  - Object is deleted somehow
  - Refer to attribute of object
  - May/may not cause crash
- Remember the 1110 rule
  - Error found != error cause
  - Cause is somewhere before
- Must work up the *call stack*
  - Part of the **binary search**



# The Challenge of Finding Errors

- *Access errors* are the hardest
  - Refer to object in memory
  - Object is deleted somehow
  - Refer to attribute of object
  - May/may not cause crash
- Remember the 1110 rule
  - Error found != error cause
  - Cause is somewhere before
- Must work up the *call stack*
  - Part of the **binary search**



- “Deletion” is not immediate
  - Marks it for deletion
  - Will be deleted later
- Can still access object
  - Data corrupted as recycled

# Primitive Confirmation Tools

---

- **Logging** (**CULog**)
  - Print out a variable value to check it
  - Alternatively print out a trace of program flow
  - **Goal**: View the internal program state
- **Assertions** (**CUAssert**)
  - Check that your assumption is true
  - Crash the code if it is not
  - **Goal**: Make error closer to the crash

# Primitive Confirmation Tools

---

- **CULog**(statement, v1, v2, v3...)
  - Uses same syntax as printf()
  - Need to use char\* to display string names
  - **Ex:** CULog("Node is %s", node->getName().c\_str())
- **CUAssert**(test, statement, v1, v2, v3...)
  - Test is any boolean statement
  - Remainder of arguments act like printf()
  - **Ex:** CUAssert(index > 0, "index is %d", index)

# Problems with Logging

---

- **Verbose**

- Code with print every animation frame
- Way too much information to sort through
- Most game designers will log to a file

- **Distortionary**

- Logging and other I/O is a blocking operation
- Will change the thread behavior of your app
- Can cause errors to appear/disappear



# Advanced Tools

---

- **Breakpoints**

- Stop the execution of the code
- Can continue running from that point
- Can continue one step at a time

- **Watches**

- Look at the value of an individual variable
- Can drill down into object attributes
- But only works when variable is in scope

# Advanced Tools

---

- **Memory Dumps**

- Look at a raw memory location
- Does not require a variable to be in scope
- Good way to look at heap for corruption

- **Thread Monitors**

- Stack traces for all running threads
- All threads are frozen by a breakpoint
- Allows you to compare state across threads

# XCode Tools

The screenshot displays the Xcode IDE interface for a project named "RocketBugs-desktop". The main editor shows the source code for `RDGameController.cpp`. A breakpoint is set at line 529, which is currently active, as indicated by the "Thread 1: breakpoint 4.1" label. The code includes comments about collision handling and a `update` method that processes key commands and applies force to a rocket.

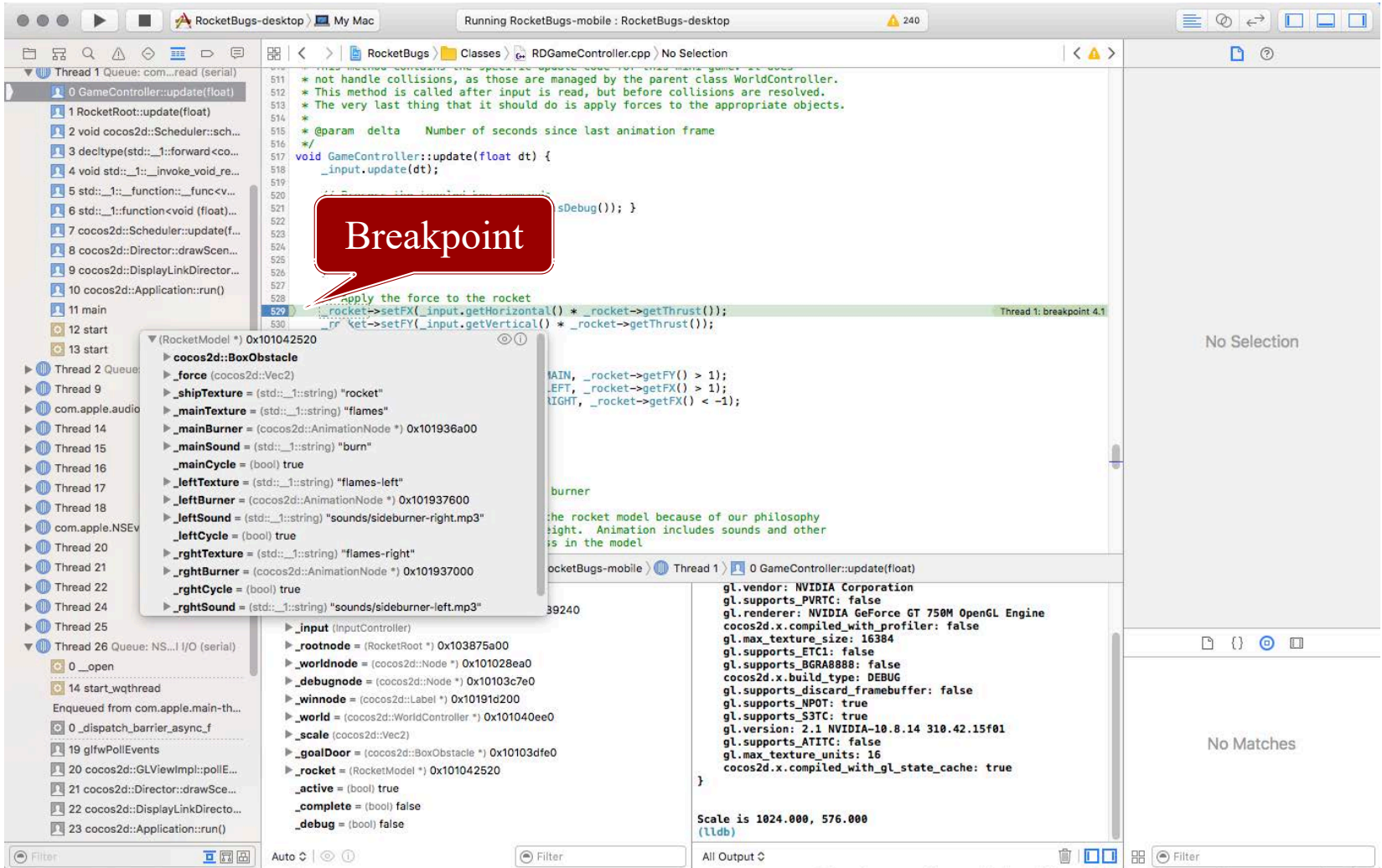
The Thread Debugger on the left shows a call stack for Thread 1, with the current frame being `0 GameController::update(float)`. A detailed view of the `(RocketModel *) 0x101042520` object is shown, listing various attributes such as `_force`, `_shipTexture`, `_mainTexture`, `_mainBurner`, `_mainSound`, `_mainCycle`, `_leftTexture`, `_leftBurner`, `_leftSound`, `_leftCycle`, `_rightTexture`, `_rightBurner`, `_rightCycle`, and `_rightSound`.

The Console window at the bottom right displays the following output:

```
gl.vendor: NVIDIA Corporation
gl.supports_PVRTC: false
gl.renderer: NVIDIA GeForce GT 750M OpenGL Engine
cocos2d.x.compiled_with_profiler: false
gl.max_texture_size: 16384
gl.supports_ETC1: false
gl.supports_BGRA8888: false
cocos2d.x.build_type: DEBUG
gl.supports_discard_framebuffer: false
gl.supports_NPOT: true
gl.supports_S3TC: true
gl.version: 2.1 NVIDIA-10.8.14 310.42.15f01
gl.supports_ATIIC: false
gl.max_texture_units: 16
cocos2d.x.compiled_with_gl_state_cache: true

Scale is 1024.000, 576.000
(lldb)
```

# XCode Tools



# XCode Tools

The screenshot displays the Xcode IDE interface for a project named "RocketBugs". The main editor window shows the source code for "RDGameController.cpp". A red callout bubble labeled "Breakpoint" points to a line of code: `rocket->setFX(_input.getHorizontal() * _rocket->getThrust());`. The left sidebar shows a list of threads and a detailed view of the "RocketModel" object, listing various attributes like textures, burners, and sounds. A green callout bubble labeled "Watches" points to the "Watches" pane at the bottom, which displays the memory dump for the selected object, including fields like `_input`, `_rootnode`, `_worldnode`, and `_rocket`. The status bar at the bottom indicates "Scale is 1024.000, 576.000 (lldb)".

```
511 * not handle collisions, as those are managed by the parent class WorldController.
512 * This method is called after input is read, but before collisions are resolved.
513 * The very last thing that it should do is apply forces to the appropriate objects.
514 *
515 * @param delta Number of seconds since last animation frame
516 */
517 void GameController::update(float dt) {
518     _input.update(dt);
519
520     // Apply the force to the rocket
521     rocket->setFX(_input.getHorizontal() * _rocket->getThrust());
522     rocket->setFY(_input.getVertical() * _rocket->getThrust());
523 }
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
```

Breakpoint

Watches

Watches

No Selection

No Matches

Scale is 1024.000, 576.000 (lldb)

# XCode Tools

The image shows a screenshot of the XCode IDE interface. The main window displays the source code for `RDGameController.cpp`. A red speech bubble labeled "Breakpoint" points to a line of code in the editor. On the left side, a blue speech bubble labeled "Thread Manager" points to the Thread Manager pane, which lists various threads and their states. Below the code editor, a green speech bubble labeled "Watches" points to the Watches pane, which shows the memory dump for the selected thread. Another green speech bubble labeled "Watches" points to the console output at the bottom of the interface.

**Thread Manager**

- Thread 1 Queue: com.apple.audio...
- 0 GameController::update(float)
- 1 RocketRoot::update(float)
- 2 void cocos2d::Scheduler::sch...
- 3 decltype(std::\_1::forward<co...
- 4 void std::\_1::invoke\_void\_re...
- 5 std::\_1::function::\_\_func<v...

**Breakpoint**

```
529 rocket->setFX(_input.getHorizontal() * _rocket->getThrust());  
530 rocket->setFY(_input.getVertical() * _rocket->getThrust());
```

**Watches**

```
(RocketModel *) 0x101042520  
  cocos2d::BoxObstacle  
    _force (cocos2d::Vec2)  
    _shipTexture = (std::__1::string) "rocket"  
    _mainTexture = (std::__1::string) "flames"  
    _mainBurner = (cocos2d::AnimationNode *) 0x101936a00  
    _mainSound = (std::__1::string) "burn"  
    _mainCycle = (bool) true  
    _leftTexture = (std::__1::string) "flames-left"  
    _leftBurner = (cocos2d::AnimationNode *) 0x101937600  
    _leftSound = (std::__1::string) "sounds/sideburner-right.mp3"  
    _leftCycle = (bool) true  
    _rightTexture = (std::__1::string) "flames-right"  
    _rightBurner = (cocos2d::AnimationNode *) 0x101937000  
    _rightCycle = (bool) true  
    _rightSound = (std::__1::string) "sounds/sideburner-left.mp3"  
  _input (InputController)  
  _rootnode = (RocketRoot *) 0x103875a00  
  _worldnode = (cocos2d::Node *) 0x101028ea0  
  _debugnode = (cocos2d::Node *) 0x10103c7e0  
  _winnode = (cocos2d::Label *) 0x10191d200  
  _world = (cocos2d::WorldController *) 0x101040ee0  
  _scale (cocos2d::Vec2)  
  _goalDoor = (cocos2d::BoxObstacle *) 0x10103dfe0  
  _rocket = (RocketModel *) 0x101042520  
  _active = (bool) true  
  _complete = (bool) false  
  _debug = (bool) false
```

**Watches**

```
gl.vendor: NVIDIA Corporation  
gl.supports_PVRTC: false  
gl.renderer: NVIDIA GeForce GT 750M OpenGL Engine  
cocos2d.x.compiled_with_profiler: false  
...  
Scale is 1024.000, 576.000  
(lldb)
```

# XCode Tools

The screenshot displays the Xcode IDE interface. On the left, a call stack is visible with the following entries:

- Thread 1 Queue: com...read (serial)
- 0 GameController::update(float)
- 1 RocketRoot::update(float)
- 2 void cocos2d::Scheduler::sch...
- 3 decltype(std::\_1::forward<co...
- 4 void std::\_1::invoke\_void\_re...
- 5 std::\_1::\_\_function::\_\_func<v...
- 6 std::\_1::function<void (float)...
- 7 cocos2d::Scheduler::update(f...
- 8 cocos2d::Director::drawScen...
- 9 cocos2d::DisplayLinkDirecto...
- 10 cocos2d::Application::run()
- 11 main
- 12 start
- 13 start

The main window shows a memory dump for address 0x101042520. A blue callout box with the text "Memory Dump" is overlaid on the dump. The dump consists of hexadecimal values and their corresponding ASCII representations.

Below the memory dump, the "Variables" pane shows the state of the selected thread (Thread 1) at the call site (0 GameController::update(float):

- this = (GameController \*) 0x103875e48
- \_assets = (cocos2d::SceneManager \*) 0x618000089240
- \_input = (InputController)
- \_rootNode = (RocketRoot \*) 0x103875a00
- \_worldNode = (cocos2d::Node \*) 0x101028ea0
- \_debugNode = (cocos2d::Node \*) 0x10103c7e0
- \_winNode = (cocos2d::Label \*) 0x10191d200
- \_world = (cocos2d::WorldController \*) 0x101040ee0
- \_scale = (cocos2d::Vec2)
- \_goalDoor = (cocos2d::BoxObstacle \*) 0x10103dfe0
- \_rocket = (RocketModel \*) 0x101042520
- \_active = (bool) true
- \_complete = (bool) false
- \_debug = (bool) false

On the right side of the Variables pane, the OpenGL context information is displayed:

- gl.vendor: NVIDIA Corporation
- gl.supports\_PVRTC: false
- gl.renderer: NVIDIA GeForce GT 750M OpenGL Engine
- cocos2d.x.compiled\_with\_profiler: false
- gl.max\_texture\_size: 16384
- gl.supports\_ETC1: false
- gl.supports\_BGRA8888: false
- cocos2d.x.build\_type: DEBUG
- gl.supports\_discard\_framebuffer: false
- gl.supports\_NPOT: true
- gl.supports\_S3TC: true
- gl.version: 2.1 NVIDIA-10.8.14 310.42.15f01
- gl.supports\_ATIIC: false
- gl.max\_texture\_units: 16
- cocos2d.x.compiled\_with\_gl\_state\_cache: true

At the bottom of the Variables pane, the scale is reported as 1024.000, 576.000 (lldb).

# Visual Studio Tools

The screenshot displays the Visual Studio IDE during a debugging session. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar shows various debugging actions like Continue, Step Over, Step Into, and Break. The status bar indicates the process is [0x37E4] JSONDemo.exe, the thread is [0x3FA8] ucrtbased.dll thread, and the stack frame is LevelModel::loadCrate.

The Memory 1 window shows a memory dump starting at address 0x02BF4B20. The dump includes hexadecimal values and their corresponding ASCII representations, such as 'Z\*...e...P+i.e.iiiiiiiiiiiiiiii...v...' and 'U.S.e.r.n.s./W.a.l.k.e.r. .W.h.i.t.e./D.e.s.k.t.o.p./B.u.g.g.y./J.S.O.N.B.u.g.s./p.r.o.j...w.i.n.3.2./D.e.b.u.g...w.i.n.3.2./j.s.o.n.s./l.e.v.e.l...j.s.o.n.../...ü«...€'.

The Source Code window shows the following code in LevelModel.cpp:

```
543 Vec2 crateSize = reader.getVec2(SIZE_FIELD);
544
545 // Get the object, which is automatically retained
546 CrateModel* crate = CrateModel::create(cratePos, (Size)crateSize);
547
548 // Using the key makes too many sounds
549 // crate->setName(reader.getKey());
550 crate->setName(reader.getString(TEXTURE_FIELD));
551 CLOG("Name is %s", crate->getName().c_str());
552 std::string btune = reader.getString(BONVTPE_FIELD);
```

The Diagnostic Tools window shows a diagnostics session of 3 seconds (3.158 s selected). It includes a timeline for Events, Process Memory (MB), and CPU Usage. A table below shows the event details:

Event	Time	Duration	Thread
Stopped at Exception	3.15s	3,158ms	[16296]

The Locals window shows the following variables:

Name	Value	Type
this	0x02bf4b20 [ _root=0x00000000 <NULL> _bounds=(origin=(x=0.00000000C LevelMo	
btype	"dynamic"	std::basic
crate	0x05f56ee8 { _crateTexture=<Error reading characters of string.> _debugCr CrateMo	
cocos2d::BoxObstacle	{ _shape=(m_centroid=(x=-1.99839716e+18 y=-1.99839716e+18) m_vertic cocos2d:	
_crateTexture	<Error reading characters of string.>	std::basic
_debugColor	{r=0xdd Y g=0xdd Y b=0xdd Y }	cocos2d:
_debugOpacity	0xdd Y	unsigned
cratePos	{x=14.50000000 y=14.25000000 }	cocos2d:
crateSize	{x=2.000000000 y=2.000000000 }	cocos2d:

The Call Stack window shows the following stack frames:

- JSONDemo.exe!LevelModel::loadCrate(cocos2d::JSONReader & reader) Line 557 C++
- JSONDemo.exe!LevelModel::load() Line 290 C++
- libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::allocate(cocos2d::Asset \* asset) Line 143 C++
- libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::loadAsync::\_l2:<lambda>() Line 125 C++
- [External Code]
- libcocos2d.dll!cocos2d::ThreadPool::threadFunc() Line 107 C++
- [External Code]
- [Frames below may be incorrect and/or missing, no symbols loaded for ucrtbased.dll]



# Visual Studio Tools

The screenshot displays the Visual Studio IDE in a debugging state. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar shows various debugging actions like Continue, Step Over, Step Into, and Break. The process being debugged is [0x37E4] JSONDemo.exe, and the current thread is [0x3FA8] ucrtbased.dll thread. The stack frame is LevelModel::loadCrate.

The Memory window shows a memory dump starting at address 0x02BF4B20. The dump includes hexadecimal values and their corresponding ASCII representations, such as "Z\*...e...P+i...e.iiiiiiiiiiiiiiiiiiii...v...".

The Code window shows the source code for LevelModel::loadCrate. A red circle highlights a breakpoint at line 550, which is annotated with a red speech bubble containing the word "Breakpoint". The code includes comments like "Using the key makes too many sounds" and "crate->setName(reader.getKey());".

The Diagnostic Tools window shows a diagnostics session of 3 seconds (3.158 s selected). It includes a timeline and a table of events. The table has columns for Event, Time, Duration, and Thread. The event "Stopped at Exception" is listed with a time of 3.15s, a duration of 3,158ms, and thread [16296].

The Locals window shows the current state of local variables:

Name	Value	Type
this	0x02bf4b20 [ _root=0x00000000 <NULL> _bounds=(origin=(x=0.00000000C LevelMo	
btype	"dynamic"	std::basic
crate	0x05f56ee8 { _crateTexture=<Error reading characters of string.> _debugCr CrateMo	
crate->BoxObstacle	{ _shape=(m_centroid=(x=-1.99839716e+18 y=-1.99839716e+18) m_vertic cocos2d:	
crate->Texture	<Error reading characters of string.>	std::basic
crate->debugColor	{r=0xdd Y g=0xdd Y b=0xdd Y }	cocos2d:
crate->debugOpacity	0xdd Y	unsigned
cratePos	{x=14.50000000 y=14.25000000 }	cocos2d:
crateSize	{x=2.000000000 y=2.000000000 }	cocos2d:

The Call Stack window shows the call sequence:

- JSONDemo.exe!LevelModel::loadCrate(cocos2d::JSONReader & reader) Line 557
- JSONDemo.exe!LevelModel::load() Line 290
- libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::allocate(cocos2d::Asset \* asset) Line 143
- libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::loadAsync::\_l2:<lambda>() Line 125
- [External Code]
- libcocos2d.dll!cocos2d::ThreadPool::threadFunc() Line 107
- [External Code]
- [Frames below may be incorrect and/or missing, no symbols loaded for ucrtbased.dll]

# Visual Studio Tools

The screenshot displays the Visual Studio IDE during a debugging session. The top menu bar includes File, Edit, View, Project, Build, Debug, Team, Tools, Architecture, Test, Analyze, Window, and Help. The toolbar shows various debugging actions like Continue, Step Over, Step Into, and Break. The status bar indicates the process is [0x37E4] JSONDemo.exe, the thread is [0x3FA8] ucrtbased.dll thread, and the stack frame is LevelModel::loadCrate.

**Memory 1** window shows a memory dump starting at address 0x02BF4B20. The dump includes hexadecimal values and their corresponding ASCII representations, such as "Z\*...e...P+i...e.ffffffffffff...v...".

The **JSONDemo** window shows the source code for `LevelModel::loadCrate`. A red circle marks a breakpoint at line 550, which is annotated with a red speech bubble containing the word "Breakpoint". The code includes comments and function calls like `Vec2 crateSize = reader.getVec2(SIZE_FIELD);` and `crate->setName(reader.getString(TEXTURE_FIELD));`.

The **Diagnostic Tools** window shows a diagnostics session of 3 seconds (3.158 s selected). It includes a timeline and a table of events. The table shows an event "Stopped at Exception" with a time of 3.15s, a duration of 3,158ms, and a thread ID of [16296].

The **Locals** window shows the current state of local variables:

Name	Value
this	0x02bf4b20 [ _root=0x00000000 <NULL> _bound...
btype	"dynamic"
crate	0x05f56ee8 { _crateTexture=<Error reading chara...
cocos2d::BoxObstacle	{ _shape=(m_centroid={x=-1.99839716e+18 y=-1.99839716e+18} m_event_cocos2d::...
_crateTexture	<Error reading characters of string.>
_debugColor	{r=0xdd Y g=0xdd Y b=0xdd Y }
_debugOpacity	0xdd Y
cratePos	{x=14.50000000 y=14.25000000 }
crateSize	{x=2.000000000 y=2.000000000 }

The **Call Stack** window shows the current call stack:

Name	Lang
JSONDemo.exe!LevelModel::loadCrate(cocos2d::JSONReader & reader) Line 57	C++
JSONDemo.exe!LevelModel::load() Line 290	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::allocate(cocos2d::Asset * asset) Line 143	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::loadAsync::_I2:<lambda>() Line 125	C++
[External Code]	
libcocos2d.dll!cocos2d::ThreadPool::threadFunc() Line 107	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded for ucrtbased.dll]	

A green speech bubble containing the word "Watches" is positioned over the Locals window.

# Visual Studio Tools

The screenshot displays the Visual Studio IDE with several key components:

- Memory Dump:** A window titled "Memory 1" showing a hex dump of memory starting at address 0x02BF4B20. A blue callout bubble labeled "Memory Dump" points to this window.
- Code Editor:** The main editor shows C++ code in `JSLevelModel.cpp`. A red callout bubble labeled "Breakpoint" points to a red dot on line 550. The code includes:

```
543 Vec2 crateSize = reader.getVec2(SIZE_FIELD);
544
545 // Using the key makes too many sounds
550 crate->setName(reader.getKey());
551 crate->setName(reader.getString(TEXTURE_FIELD));
552 CLOG("Name is %s", crate->getName().c_str());
553 std::string btune = reader.getString(BODYTYPE_FIELD);
```
- Diagnosics Tools:** A window on the right showing diagnostic tools, including a timeline and a table of events. The table shows:

Event	Time	Duration	Thread
Stopped at Exception	3.15s	3,158ms	[16296]
- Locals:** A window at the bottom left showing local variables for the current function:

Name	Value
this	0x02bf4b20 [/_root=0x00000000 <NULL> _bound...
btype	"dynamic"
crate	0x05f56ee8 [_crateTexture=<Error reading chara...
cocos2d::BoxObstacle	[_shape=(m_centroid=(x=-1.99839716e+18 y=-1.99...
_crateTexture	<Error reading characters of string.>
_debugColor	{r=0xdd Y g=0xdd Y b=0xdd Y}
_debugOpacity	0xdd Y
cratePos	{x=14.50000000 y=14.25000000}
crateSize	{x=2.00000000 y=2.00000000}
- Call Stack:** A window at the bottom right showing the call stack:

Name	Lang
JSONDemo.exe!LevelModel::loadCrate(cocos2d::JSONReader & reader) Line 57	C++
JSONDemo.exe!LevelModel::load() Line 290	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::allocate(cocos2d::Asset * asset) Line 143	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::loadAsync::_l2:<lambda>() Line 125	C++
[External Code]	
libcocos2d.dll!cocos2d::ThreadPool::threadFunc() Line 107	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded for ucrtbased.dll]	

# Visual Studio Tools

The screenshot displays the Visual Studio IDE with several debugging tools open:

- Memory Dump:** A window showing a memory dump at address 0x02BF4B20. A blue callout bubble labeled "Memory Dump" points to this window.
- Breakpoint:** A red dot on line 550 of the code indicates a breakpoint. A red callout bubble labeled "Breakpoint" points to this dot.
- Watches:** A window showing the state of local variables. A green callout bubble labeled "Watches" points to this window.
- Call Stack:** A window showing the sequence of function calls. A blue callout bubble labeled "Call Stack" points to this window.

The code in the background shows a function `loadCrate` in `JSLevelModel.cpp` that reads JSON data and creates a `CrateModel` object. The `crate` object is highlighted in the Watches window.

```
543 Vec2 crateSize = reader.getVec2(SIZE_FIELD);
544
545 // Using the key makes too many sounds
550 crate->setName(reader.getString(TEXTURE_FIELD));
551 CLOG("Name is %s", crate->getName().c_str());
std::string btune = reader.getString(BODYTYPE_FIELD);
```

Name	Value
this	0x02bf4b20 [/_root=0x00000000 <NULL> _bound...
btype	"dynamic"
crate	0x05f56ee8 [_crateTexture=<Error reading chara...
cocos2d::BoxObstacle	[_shape=(m_centroid=(x=-1.99839716e+18 y=-1.95...
_crateTexture	<Error reading characters of string.>
_debugColor	{r=0xdd Y g=0xdd Y b=0xdd Y}
_debugOpacity	0xdd Y
cratePos	{x=14.50000000 y=14.25000000}
crateSize	{x=2.00000000 y=2.00000000}

Name	Lang
JSONDemo.exe!LevelModel::loadCrate(cocos2d::JSONReader & reader) Line 557	C++
JSONDemo.exe!LevelModel::load() Line 290	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::allocate(cocos2d::Asset * asset) Line 143	C++
libcocos2d.dll!cocos2d::GenericBaseLoader::Coordinator::loadAsync::_I2:<lambda>() Line 125	C++
[External Code]	
libcocos2d.dll!cocos2d::ThreadPool::threadFunc() Line 107	C++
[External Code]	
[Frames below may be incorrect and/or missing, no symbols loaded for ucrtbased.dll]	

# Visual Studio Tools

The screenshot displays the Visual Studio IDE with several debugging tools open:

- Memory Dump:** A window showing a memory dump at address 0x02BF4B20. A blue callout bubble labeled "Memory Dump" points to this window.
- Breakpoint:** A red circle on the code editor indicates a breakpoint. A red callout bubble labeled "Breakpoint" points to it.
- Watches:** A window showing local variables and their values. A green callout bubble labeled "Watches" points to this window.
- Call Stack:** A window showing the current call stack. A blue callout bubble labeled "Call Stack" points to it.
- Diagnostic Tools:** A window showing diagnostic information, including a timeline and process memory usage. A blue callout bubble labeled "Threads have a separate window" points to this window.

The code editor shows the following code snippet:

```
543 Vec2 crateSize = reader.getVec2(SIZE_FIELD);  
544  
545 // Using the key makes too many sounds  
546 // crate->setName(reader.getKey());  
547 crate->setName(reader.getString(TEXTURE_FIELD));  
548 CLOG("Name is %s", crate->getName().c_str());  
549 std::string btune = reader.getString(BODYTYPE_FIELD);
```

# Breakpoint Strategies

---

- **Break early**

- Break before the error, to check everything is okay
- Step forward and watch how the code changes

- **Break infrequently**

- If you always break, cannot initialize or animate anything
- Design special conditionals for your breakpoint

- **Break on deletion**

- Put breakpoints inside of all your destructors
- Allows you to track accidental deletion

# Problems with Code Stepping

---

- Code stepping is not “thread safe”
  - Will never leave your current thread
  - Have to choose “continue” instead of “step”
- Makes it very difficult to find thread errors
  - May miss when a variable changes state
  - We had many problems in an old AudioEngine
- **Solution:** Rely heavily on assertions
  - Assert every variable shared across threads
  - Assert them everywhere they may change

# Case Study: JSON Loading

---

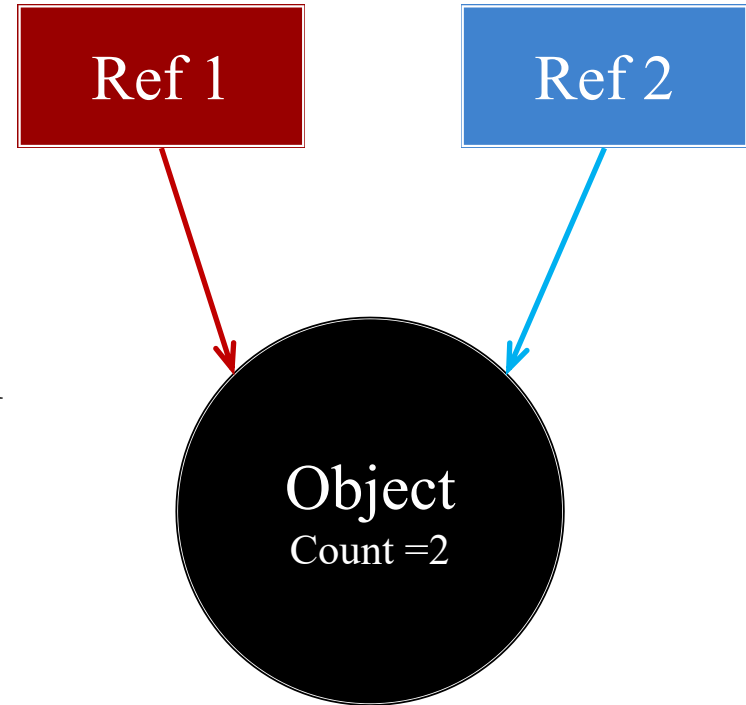
- Problem in Cocos2d-x, an older engine
  - Not a C++11 compliant engine
  - Did not support smart pointers (or anything)
  - Instead all game objects had *reference counting*
- Manual reference counting leads to mistakes
  - Only slightly better than manual deletion
  - Even Apple has abandoned this in Objective-C
- But very instructive for **debugging memory**



# Aside: Reference Counting

---

- Every object has a **counter**
  - Tracks number of “owners”
  - No owners = memory leak
  - Increment when get reference
- Often an explicit method call
  - Historically called `retain()`
- Decrement when reference lost
  - Method call is `release()`
  - If makes count 0, delete it



# Scene Graphs the Old Way

---

// create a new instance

Node\* node = Node::create();

Raw pointers!!

**node->retain();**

Manual refence counts!!

// Add the node to scene graph

scene->addChild(**node**);

// Release the local reference

**node->release();**

// Remove from scene graph

scene->removeChild(**node**);

# Scene Graphs the Old Way

---

// create a new instance

Node\* node = Node::create();

Custom allocator

**node->retain();**

Reference count 1

// Add the node to scene graph

scene->addChild(**node**);

Reference count 2

// Release the local reference

**node->release();**

Reference count 1

// Remove from scene graph

scene->removeChild(**node**);

Reference count 0

node is **deleted**

# Scene Graphs the Old Way

---

// create a new instance

Node\* node = Node::create();

Custom allocator

**node->retain();**

Reference count 1

// Add the node to scene graph

scene->addChild(**node**);

Reference count 2

// Do not release the local reference

// Remove from scene graph

scene->removeChild(**node**);

Reference count 1

**Memory Leak!**

# Case Study: JSON Loading

---

- Problem was a thread *race condition*
  - Appeared on Windows, but not MacOS
  - Because of particular Windows thread schedule
  - But technically unsafe on all platforms
- Found by putting **breakpoints in destructors**
  - Models getting deleted immediately after creation
  - Watched the reference counts to find problem
  - There was a stray `release()` before `retain()`

# Case Study: b2BlockAllocator

---

- *Memory address* problem in Box2D engine
  - Problem was because we put Box2D in a DLL
  - Required stepping through the allocation process
  - Required **memory dumps** to view the heap
- Problem with the *static global variables*
  - DLLs have a distinct global space
  - BlockAllocator was initialized inside of the DLL
  - When it was used outside the DLL, not initialized

# Summary

---

- Two main strategies to debugging
  - **Confirmation:** Make sure code does what you think
  - **Binary Search:** Find where confirmation wrong
- Primitive tools in code on all platforms
  - Logging with CULog
  - Assertions with CUAssert
- Advanced tools in professional IDEs
  - Breakpoints and Watches
  - Thread Monitors (to see call stack)
  - Memory Dumps