Shining Force PC Code Overview
December 2010 Edition

kenhilf@comcast.net

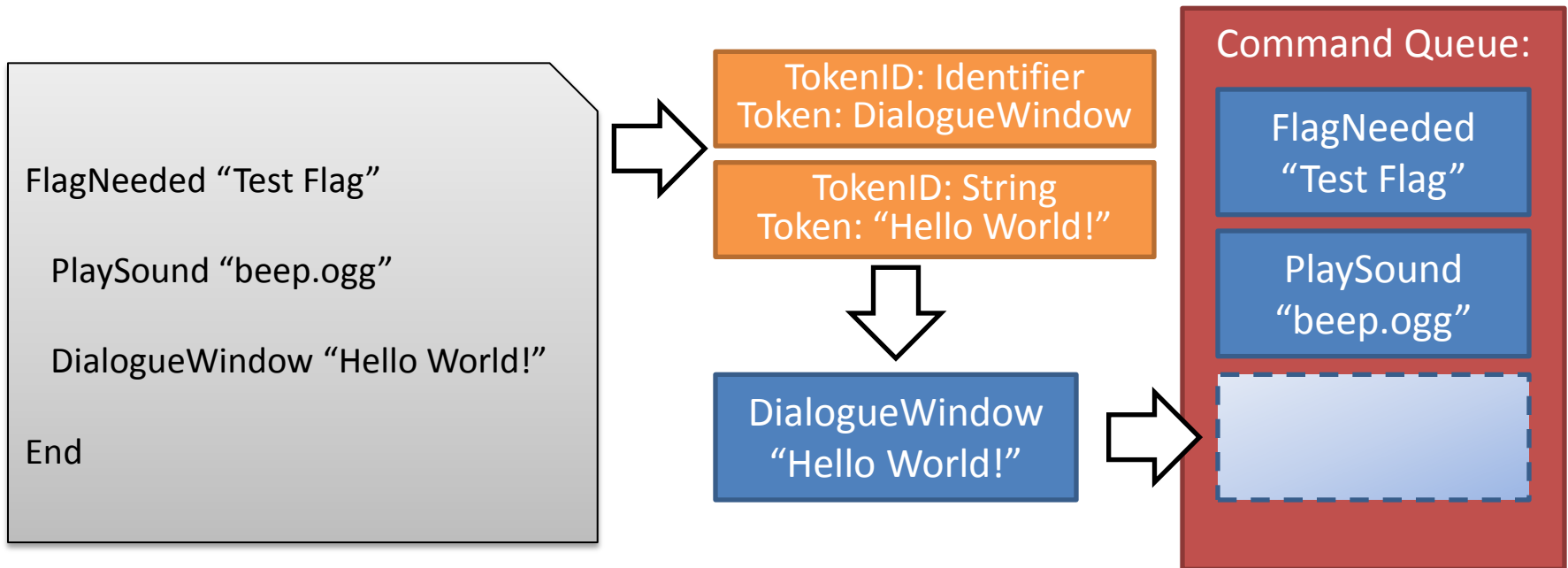http://kenhilfportfolio.blogspot.com

Scripting Engine
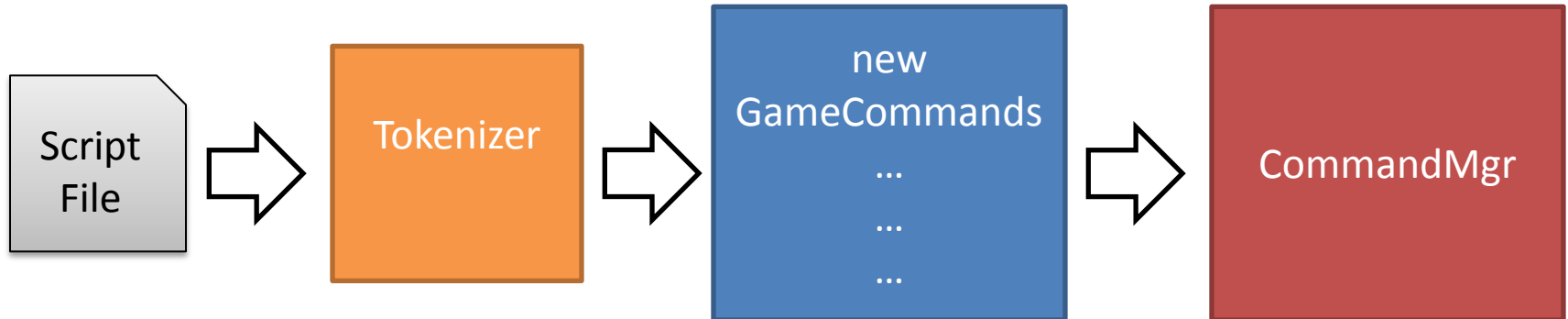
- Content creation system for SFPC game engine.
- Uses custom Tokenizer class to read in script files and turn them into Tokens from which GameCommands are built.
- GameCommands are then added into the CommandMgr's queue of commands which are executed during the CommandMgr's Update() pass.

Scripting Engine

FlagNeeded "Test Flag"

PlaySound "beep.ogg"

DialogueWindow "Hello World!"

End

TokenID: Identifier
Token: DialogueWindow

TokenID: String
Token: "Hello World!"

DialogueWindow
"Hello World!"

Command Queue:

FlagNeeded
"Test Flag"

PlaySound
"beep.ogg"

• Tokenizer reads in all the commands in a given script file and turns them into Tokens.
• GameCommands are built from Tokens and are then pushed to the back of the CommandMgr's command queue.

CommandMgr::Update()

```
while (m_pCurrCommand)
{
   if (m_pCurrCommand->Update(ticks))
   {
      delete m_pCurrCommand;
      m_pCurrCommand = NULL;

      if (m_comQueue.size() > 0)
      {
         m_pCurrCommand = m_comQueue.front();
         m_comQueue.pop();
      }
   }
   else break;
}
```

• m_pCurrCommand is a GameCommand pointer, the virtual Update() fn is called on the actual type of GameCommand it points to through polymorphism.

• As long as the active command returns true, more commands will be run from the queue until none are left or a command returns false.

• A command that returns false is either waiting for something to occur or has a built in delay. It will accumulate ticks until the specified time, then execute and return true, signaling the CommandMgr to proceed to the next command.

GameCommands

```
// Base GameCommand class
class GameCommand
{
public:
 GameCommand(const CommandID::Type id);
 virtual ~GameCommand();
 CommandID::Type GetID();
 virtual bool Update(const double ticks);

protected:
   CommandID::Type m_commandID;
};
```
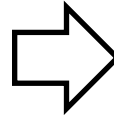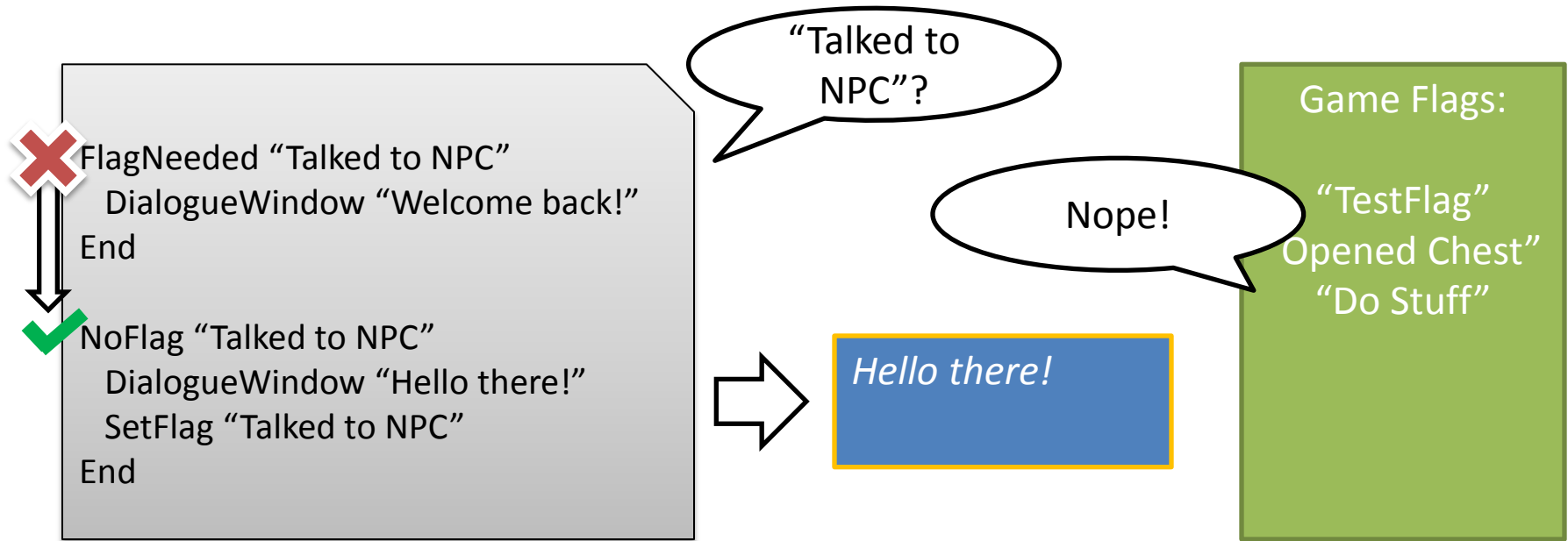
```
// PlaySound - Plays a sound effect once
class GCPlaySound : public GameCommand
{
public:
 GCPlaySound(const std::string& filename);
 virtual ~GCPlaySound();
 virtual bool Update(const double ticks);

private:
   std::string m_filename;
};
```

- GameCommand is a is a simple base class that all scripting commands inherit from.
- All commands execute their actions in the Update function.
- Ex - When a PlaySound's Update() fn is run, it calls AudioMgr::PlaySoundEffect() with the given filename and the requested sound is played.

Scripting Simple Game Logic

"Talked to NPC"?

FlagNeeded "Talked to NPC"
    DialogueWindow "Welcome back!"
End

NoFlag "Talked to NPC"
    DialogueWindow "Hello there!"
    SetFlag "Talked to NPC"
End

Nope!

Game Flags:

"TestFlag"
"Opened Chest"
"Do Stuff"

Hello there!

- Flag system allows script writers to make their own variables.
- In this example script, the first time the script is run, the flag "Talked to NPC" has not been set yet. The first block is skipped since the flag must exist for those commands to run.
- Instead the second block gets executed. "Hello there!" gets displayed and the flag is set by the SetFlag command.

Scripting Simple Game Logic
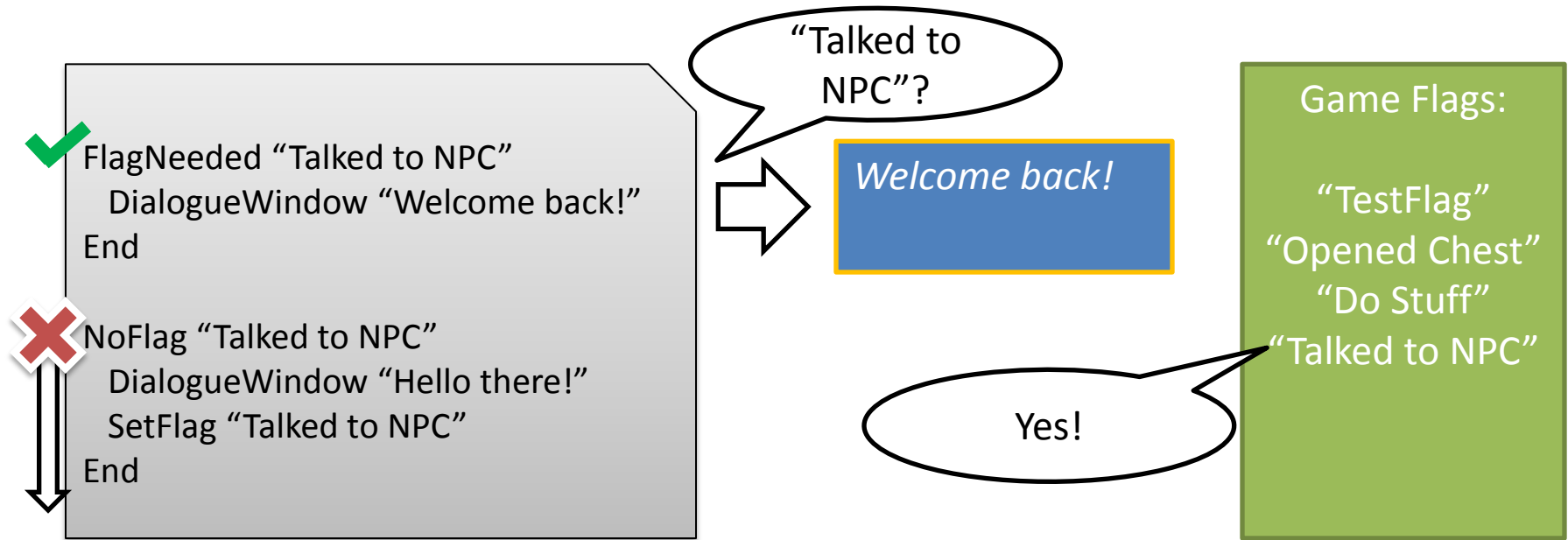
- The second time the script is run, the flag "Talked to NPC" has already been set. The first block is run and the text window with "Welcome back!" is displayed.
- As the second block is only run if no flag exists with the name "Talked to NPC", it gets skipped over.

Items and Actors

```
BeginNewItem "Mage Staff"

    SetItemClass "Weapon"
    SetItemIcon "Mage Staff"
    SetCutsceneGfxFilename "None"
    SetItemStats 0 0 27 0 0 0
    SetItemGoldValue 6300
    AddItemRange 1
    AddItemEquipable "SORC"
    AddItemEquipable "WIZ"

EndNewItem
```

```
BeginNewActor "BOWIE" "Player"

    SetActorClass "SDMN"
    SetActorMobility "Human"
    SetActorStats 1 12 8 0 6 4 4 6
    SetActorTileset "sfsdmn.bmp"
    SetActorFaceset "face_bowie.bmp"
    SetActorBattleset "battle_bowie.bmp" 5
    SetActorPosition 52 49
    SetActorFacing "Down"

# Optional Fields, SetActorSpell and SetActorItem Only!
    SetActorSpell "Egress"
    SetActorItemEquipped "Wooden Sword"

EndNewActor
```
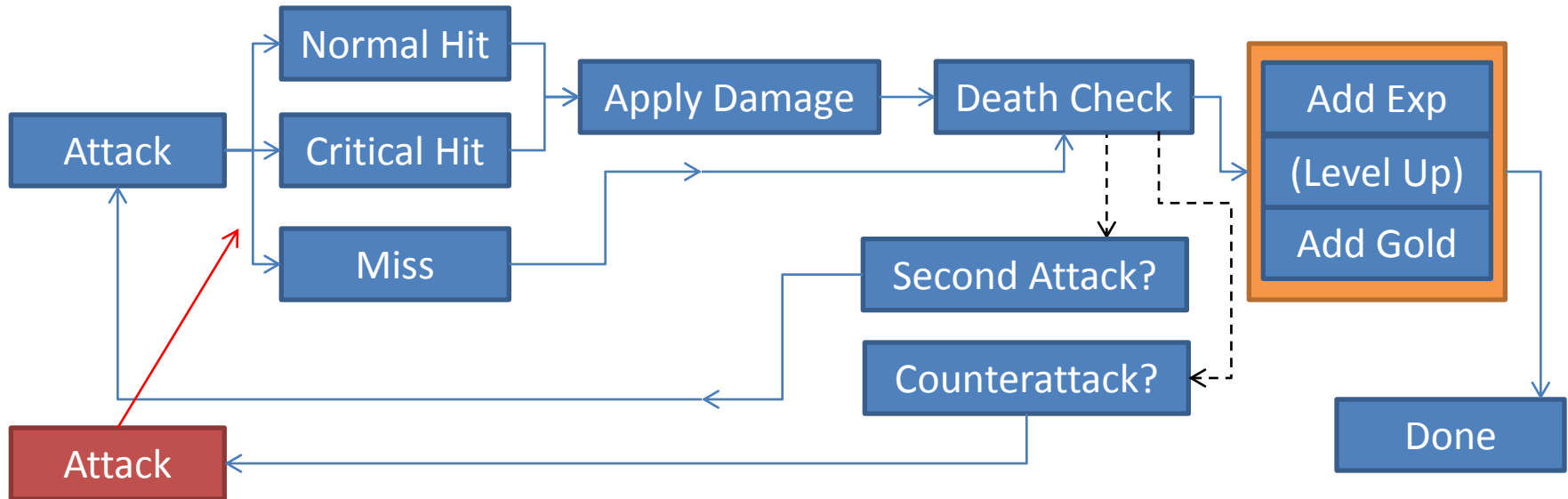
• In addition to game events, both Items and Actors are now built using the scripting engine.

• As these classes will grow as work on the engine continues, new script commands can be easily built to make their new properties user configurable as well.

SFPC – Combat System

• Combat cutscenes in SFPC use a Finite State Machine system, with CombatEvents using a polymorphism model similar to that used by the CommandMgr and GameCommands.

• Unlike GameCommands, CombatEvents usually contain their own logic and will queue up other CombatEvents based on their own dice rolls and the rules of combat flow.