# Artificial Intelligence - Modularity with Behavior Trees

Andy Vuong
*Grand Valley State University*

# Artificial Intelligence - Modularity with Behavior Trees

Andy Vuong

A Project Submitted to

**GRAND VALLEY STATE UNIVERSITY**

In

Partial Fulfillment of the Requirements

For the Degree of

**Master of Science in Applied Computer Science**

**School of Computing and Information Systems**

**December 2020**

GRAND VALLEY
STATE UNIVERSITY

The signatures of the individuals below indicate that they have read and approved the project of Andy Vuong in partial fulfillment of the requirements for the degree of Master of Science in Applied Computer Science.

Robert Adams, Project Advisor Date

Robert Adams, Graduate Program Director Date

Paul Leidig, Unit head Date

## Abstract

Artificial intelligence (AI) is a growing field of interest in computer science and is becoming increasingly important. There are numerous applications of AI with one of the most common ones being in video games. AI in video games often requires complex behavior for non-player characters (agents) within the game (e.g., enemies). One challenge for video game developers is to provide robust agent behavior while reducing duplication of code. Agent behavior often consists of a choice between independent actions ("should I run or shoot?"), but good design principles dictate that the code for those actions should not be duplicated in each agent. As more behaviors and actions are introduced, more code for common actions are repeated in each one, thus ballooning the code. One AI technique to combat this problem is behavior trees. Behavior trees allow behaviors to be composed from independent actions, while reducing the repetition of code. Modularity is also another benefit of behavior trees as behaviors can be created through different combinations of independent actions. The purpose of this project is to explore the concept of behavior trees and determine its effectiveness to the repetition issue. A simple space-shooter type game is used to show the application of behavior trees and how effective it is when used to create different types of behaviors for the AI. The result demonstrates that behavior trees can be used to not only create different types of complex behaviors for the AI, but also reduce the repetition of code through reusability and modularity.

## Introduction

Artificial intelligence (AI) is a growing field in technology that is becoming more important as time goes on. One common application of AI is for video games to allow players to have an enemy to face in cases where another player is infeasible, unavailable, and/or unsuitable. The complexity of the AI can vary depending on the type of game. Simple AI's can be created for games such as Tic-Tac-Toe while more advanced AI is required for complex games such as chess. As AI grows in complexity, one of the common issues is the ballooning of code due to repetition. An example can be seen in a simple 2-D shooter-type game where enemies may behave differently given their environment, yet they all share common actions such as walking and/or shooting. For a few types of enemies, repetition may not be a significant issue but when a large amount of different behaviors are needed, common code for simply walking and/or shooting is repeated for each of these behaviors. One design technique that mitigates this issue is behavior trees. Behavior trees allow for the creation of complex behaviors through modularity while also minimizing the repetition of code. One of the goals of this project is to explore the application and effectiveness of behavior trees for the development of AI in video games.

The main motivation for this project is to explore game development with the programming language C++. The language of choice is the result of some research on what languages are being used in the game development area, and C++ was mentioned as a common language. The library of choice is SDL 2 and the main reasoning behind this decision was due to its ease of use. SDL and another library called SFML were two common libraries mentioned during research and SDL was chosen mainly due to its widespread adoption within the game development field. This would allow for examples and assistance to be found easier when needed. As a result, learning the language is another motivator for this project.

## Project Management

Project management was straightforward if a bit simple. Weekly goals or sprints were the main approach with time spent every day reaching those goals. Weekly meetings with my advisor were also critical as they helped the project stay on track consistently with goals to reach for each meeting. The goals were further divided into sub-goals to accomplish for a day to help divide the goal of the week into smaller tasks.

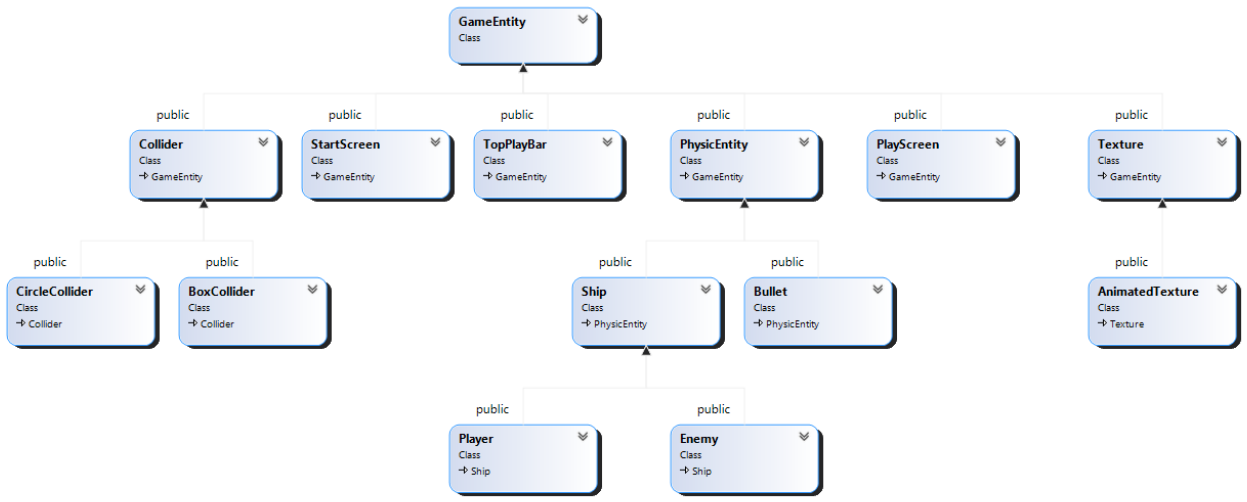
The project was worked on during the weekdays and about 3-5 hours per day were spent on the project, with variations depending on external factors. This amount of time allowed for some work to be done on some basic game features of the project and improved upon while the main goal of the project was being focused on. The allotted time also allowed for some slack when progress was a bit off-schedule and also allowed for any spare time to be used on fixing the bugs of the project.

A GitHub repository was used to help store and keep track of the project's progress each week. The link to the repository is located in the appendix. The repository was updated weekly when possible to coincide with the weekly meetings. This also helped synchronize my workstations so I could be better prepared to showcase the project currently. Some updates were delayed as work was still being done on the project and it was not presentable in the state it was in so the update was pushed back until it was in a presentable state.

## Organization

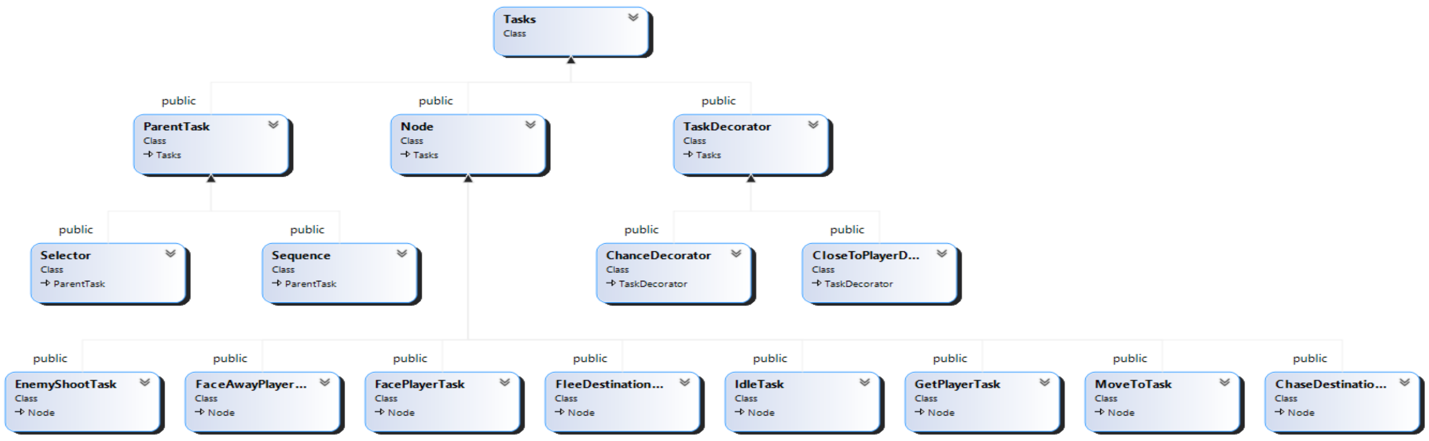
The architecture of the project is a bit complex due to the involvement of AI and the behavior tree. The main components of the behavior tree were created with the assistance of some examples found on the internet [2]. These examples helped craft the basic architecture while custom components were added where necessary. The tasks for the AI to perform were divided into smaller basic tasks. One example is having the AI flee from the player where the task is broken up into smaller tasks, in this case a task to get the player's position, a task to calculate the vector of where the AI should move to, and a task to perform the movement itself. These smaller tasks demonstrate the modularity component of the behavior trees as reusing each task and combining them with other tasks leads to different types of behaviors.

SDL 2 was the main library of choice and a simple game engine needed to be developed using its components. Due to the limited timeframe I had, I used a series of video tutorials by a person named Ather Omar to help accelerate the learning process [1]. The basic skeleton of a game engine was based on the tutorial and their other tutorials helped in adding more advanced features to the engine. The game engine itself contained the basic components such as an audio manager, a screen manager, a sprite class, etc. to get a simple game up and running. The "GameEntity" class acted as a base class that many other classes inherited from as it contained basic functionality shared among each other. Functions such as "SetPosition", "SetScale" and "Rotate" are some of the common functions used among all the classes that inherit from the "GameEntity" class.



*Class diagram for the GameEntity class*

The behavior tree component of the project was based on an example found on a blog that detailed what a behavior tree was [2]. This served as the skeleton for the behavior tree while additional components were added as necessary. The behavior tree consisted of many components that were based on some abstract classes. The main abstract classes were “Tasks”, “ParentTask”, and “TaskController” to name a few. The “Tasks” class was used to represent a simple task or action to perform, such as moving the AI, calculating where the AI should move to, and/or setting the direction of the AI. “ParentTask” was a class that consisted of multiple tasks and kept track of what the current task is while “TaskController” was simply a controller type class that kept track of the status of the task it was attached to. Classes such as the “Selector” inherited from the “ParentTask” class and added the functionality of only executing a single task out of the many it contained. The “Sequence” class also inherited from the “ParentTask” class and executed all the tasks it had in a sequential order. Finally, a “TaskDecorator” abstract class was used to decorate some tasks to add additional functionality, most of which were conditionals. An example is the “ChanceDecorator” class which added the condition that the task would only be executed if the random number that was generated was greater than or equal to a threshold. These are some of the components that make up the behavior tree for the project and allow the AI to make decisions based on its environment and on chance. Below is a basic class diagram of the components for the behavior tree and the decorator components.



*Class diagram of the major components for the behavior tree*

## Reflection

The overall goal of learning C++ from this project was quite a success as I learned a lot about C++, such as multiple inheritance and smart pointers. Learning efficient memory management was also useful in taking full advantage of the lower memory footprint that C++ offers. Creating a simple game engine was a quick learning process due to the help of tutorial videos and the belief that I had to rush to the game part as quickly as possible so I can begin work on the AI and the behavior trees. As a result, I believe I could have gotten a better understanding of some concepts such as sprite movement via vectors. This would have helped later down the road as I struggled with understanding how to calculate the direction the AI should be facing and where the AI should move to next. Learning some of the intricacies of C++ would have also helped me as I ran into circular dependency troubles later during the project as I was unfamiliar with it.

Quite a bit of knowledge was gained from this project. Learning C++ and the benefits of it was quite useful and gaining some experience in game development was also a plus. Artificial intelligence was also another field that I had minimal experience in so learning the design process and how to develop an AI was very useful and interesting. Overall, the goal of this project was achieved as I learned quite a bit of C++ as well as learning about behavior trees. Quite a bit of insight was gained on the different approaches to developing behaviors for AI and through behavior trees, I learned about the complexity of creating AI for a simple video game. The end result was a simple 2D space-shooter demonstrating the application of behavior trees and the knowledge I acquired for game and AI development.

## Conclusion

A simple basic space shooter type game was built with basic features. The game is functional and has very basic gameplay. The main goal of this game was to demonstrate the benefits of behavior trees and how they could solve the issue of modularity and repetition. To accomplish this, a basic behavior tree was created for the game’s AI and basic behaviors for the enemy were built from it.

Behavior trees served as a good solution to the problem of modularity and repetitive code. The implementation within the space shooter project helped create unique behaviors for different types of enemies while minimizing repetitive code through reuse. Having common actions such as movement and shooting become task nodes allows the game to reuse that code for each behavior rather than have that code be repeated for each behavior. A basic behavior tree allowed some unique behaviors for the AI such as a sniper-like behavior where the AI flees to a safe distance before shooting. With the reuse of nodes, new behaviors can be created by reorganizing the task nodes and/or adding in decorators that apply conditions.

A basic behavior tree was implemented for the project, but a more advanced version could be implemented that could allow for more complex behaviors. The game itself is also very basic and could be made to be more complex by implementing features such as line-of-sight (LoS), With more advanced features, this would increase the amount of behaviors possible to create through a behavior tree. Future studies could also explore the decision-making side of behavior trees as the decorators are limited to simple conditions. This limitation can prove to be a roadblock for some complex behaviors and alternative solutions can be explored.

## Appendices

*Repository*: https://github.com/VectorConvoy/SimpleShooterV1

### Resources

1. https://www.youtube.com/c/AtherOmar/featured

   User who created a series of tutorials that were used to build the skeleton of the game engine as well as learning SDL

2. http://magicscrollsofcode.blogspot.com/2010/12/behavior-trees-by-example-ai-in-android.html

   A blog post from a user who detailed their behavior tree and how it works from their game. This was used as a guide, example, and skeleton for the construction of the behavior tree.

3. https://opengameart.org/content/space-game-starter-set

   Assets used in the construction of the game. Link includes the  player sprite, enemy sprite, explosion sprite, and explosion audio clip.

4. https://opengameart.org/content/boxy-bold-font-split

   Font used for text displayed in the game.

5. https://opengameart.org/content/pixel-fonts-by-pix3m

   Another font type used for the text displayed in the game

6. https://opengameart.org/content/game-over-theme

   Music used during a game over screen

7. https://opengameart.org/content/whoosh-1

   SFX used for the game

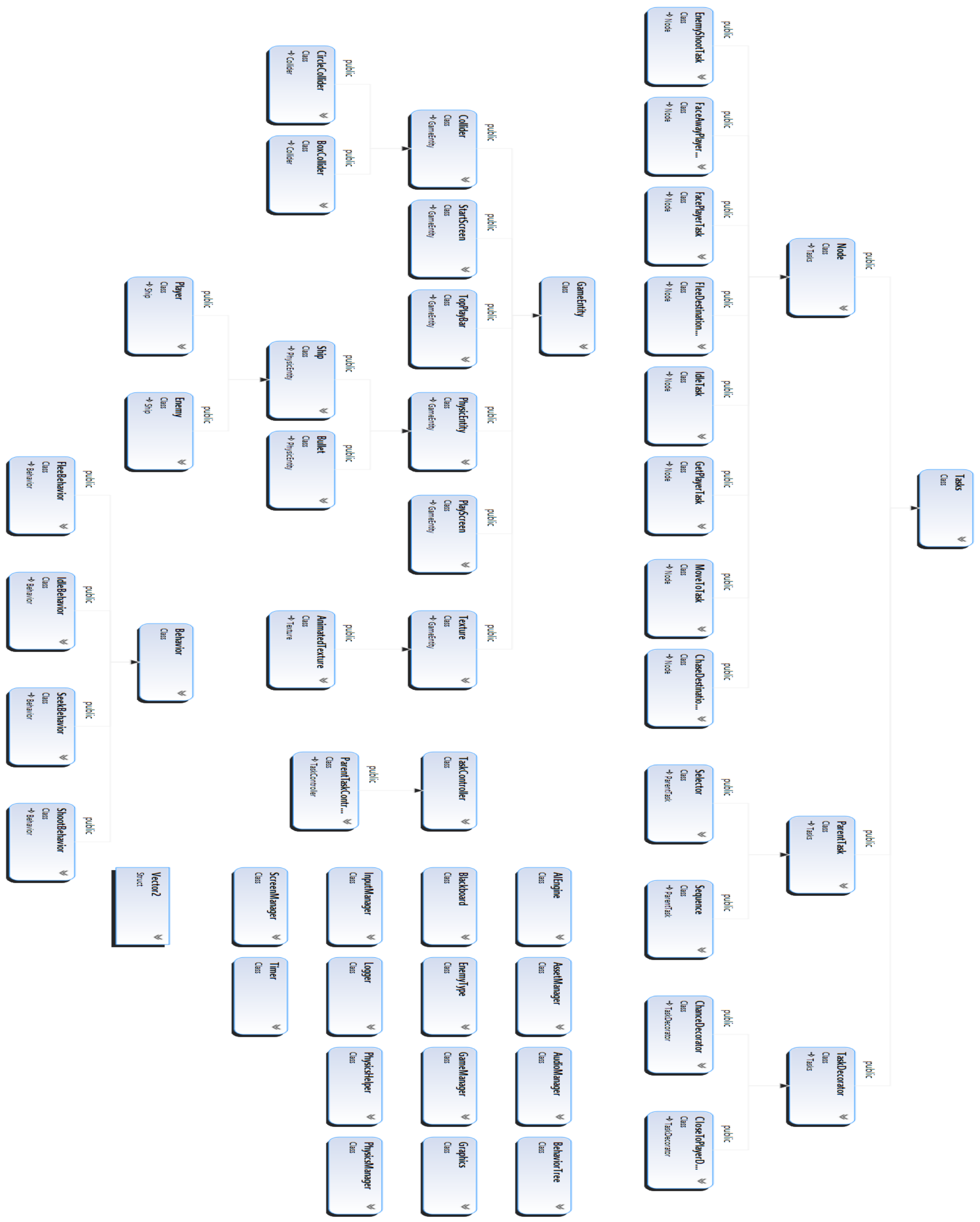
8. https://opengameart.org/content/health-bar

   Sprites used for displaying player's health

9. https://opengameart.org/content/battle-midi

   Music used in the background.

## Full Class Diagram



EnemyShootTask
FaceAwayPlayer...
FacePlayerTask
FleeDestination...
IdleTask
GetPlayerTask
MoveToTask
ChaseDestinatio...
Node
Tasks
Selector
Sequence
ParentTask
ChanceDecorator
CloseToPlayerD...
TaskDecorator
CircleCollider
BoxCollider
Collider
StartScreen
TopPlayBar
GameEntity
PhysicsEntity
PlayScreen
Texture
AnimatedTexture
Player
Enemy
Ship
Bullet
FleeBehavior
IdleBehavior
SeekBehavior
ShootBehavior
Behavior
ParentTaskContr...
TaskController
Vector2
Struct
ScreenManager
Timer
InputManager
Logger
PhysicsHelper
PhysicsManager
Blackboard
EnemyType
GameManager
Graphics
AIEngine
AssetManager
AudioManager
BehaviorTree