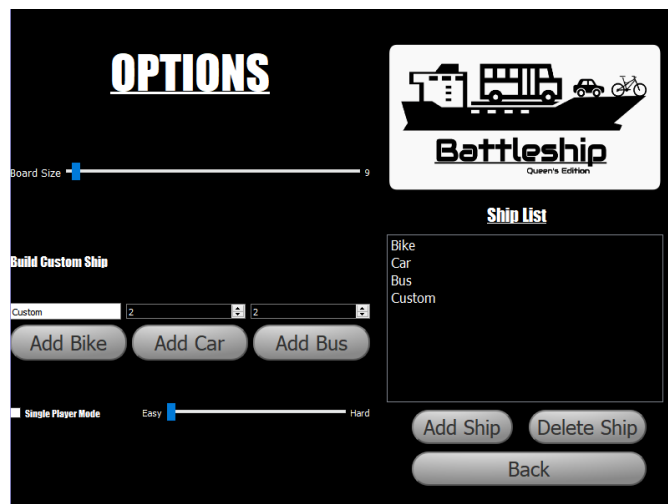


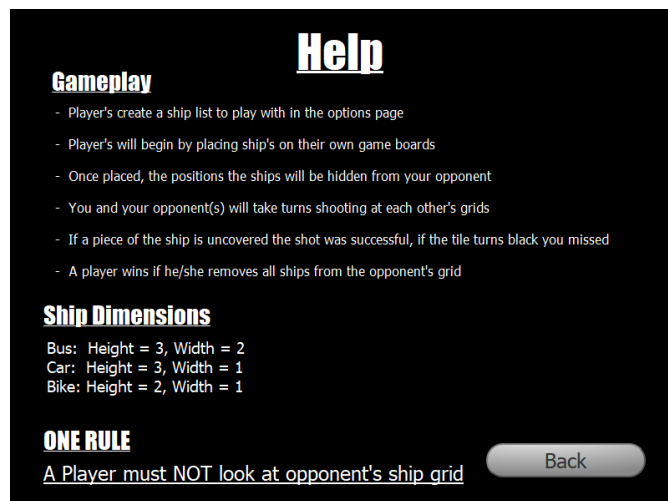
## Main Menu Window



Options Window



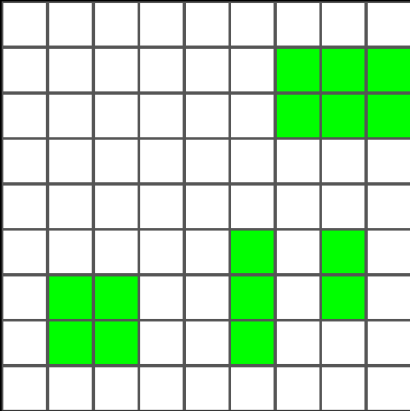
Help Window



## Ship Placement Window

(One Car, Bus, Bike, and Custom Ship have been placed)

### Ship Placement



**Ships:**

**Choose Direction:**  
If not selected, Vertical is assumed

■ Horizontal

**Turn Indicator**

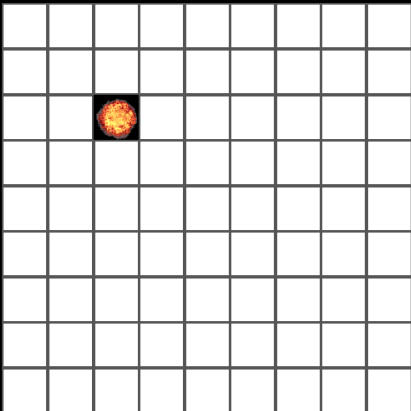
**P1**

Back

Done

## Shot Window

### Shooting Screen



**Sunk Ships**

Bikes Sunk: 0

Cars Sunk: 0

Buses Sunk: 0

Custom Ships Sunk: 0

**Turn Indicator**

**P1**

**Player Stats**

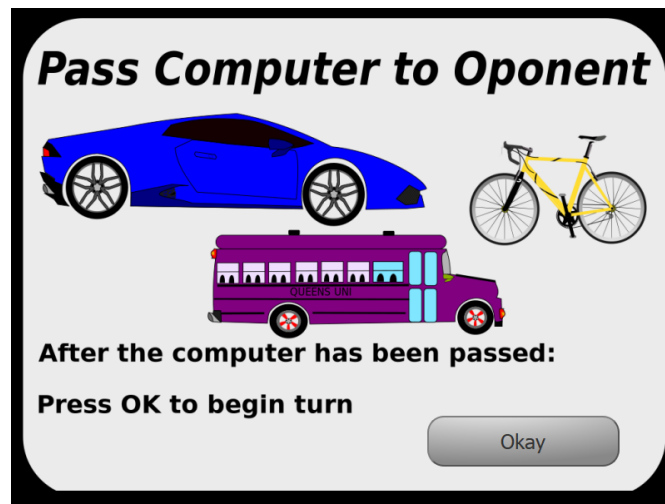
Total Shots 1

Total Hits 1

Shot Accuracy 100 %

End Turn

## Pass to Opponent Window



### Win Window



## Testing

### Frontend/State Based Testing

In order to test the frontend, we played our game! We would each launch our app on our separate laptops and visit the different pages while specifically looking for bugs. We found this was the most effective way to test the frontend as bugs in the backend and frontend can become extremely apparent while looking at the app from a user's perspective rather than only looking at the source code.

### Backend

The backend tests were written in the Cobol/FrontEndMain/TestBattleShip/TestBattleShip directory using the QT Test framework. We unit tested our backend which is mainly the Grid.cpp and Ship.cpp classes. We made sure to test functions with different combinations of parameters so that we could ensure that our backend was robust.

We didn't use write any custom hierarchical classes so we didn't conduct any polymorphism testing.

*Sample output from running test suite:*

```

21:36:47: Starting /Users/daniellemott/Desktop/CISC320/cobol/FrontEndMain/TestBattleShip/build-TestBattleShip-Desktop_Qt_5.15.1_clang_64bit-Debug/TestBattleShip.app/Contents/MacOS/TestBattleShip ...
***** Start testing of TestBattleShip *****
Config: Using QTest library 5.15.1, Qt 5.15.1 (x86_64-little_endian-lp64 shared (dynamic) release build; by Clang 10.0.0 (clang-1000.11.45.5) (Apple))
PASS : TestBattleShip::initTestCase()
PASS : TestBattleShip::test_default_ship_constructor()
PASS : TestBattleShip::test_custom_ship_constructor()
PASS : TestBattleShip::test_custom_ship_constructor_exceptions()
PASS : TestBattleShip::test_place_ship()
PASS : TestBattleShip::test_ship_get_health()
PASS : TestBattleShip::test_ship_hit()
PASS : TestBattleShip::test_ship_overloaded_equals()
PASS : TestBattleShip::test_ship_copy_constructor()
PASS : TestBattleShip::test_custom_grid_constructor()
PASS : TestBattleShip::test_grid_exception()

HIT

MISS

HIT

PASS : TestBattleShip::test_grid_shoot()
PASS : TestBattleShip::test_grid_shoot_exception()

HIT

HIT

HIT

HIT

PASS : TestBattleShip::test_grid_sunk()
PASS : TestBattleShip::cleanupTestCase()
Totals: 15 passed, 0 failed, 0 skipped, 0 blacklisted, 3ms
***** Finished testing of TestBattleShip *****
21:36:47: /Users/daniellemott/Desktop/CISC320/cobol/FrontEndMain/TestBattleShip/build-TestBattleShip-Desktop_Qt_5.15.1_clang_64bit-Debug/TestBattleShip.app/Contents/MacOS/TestBattleShip exited with code 0

```

## Integration Testing

Integration testing was essentially completed whenever any code was merged into the repository (essentially bi-weekly/weekly). Each developer that pushed code into the repository was responsible for ensuring that it didn't break the existing code in the repository. We essentially followed the 'top down' approach as mentioned in Lecture 41. Since we're building a GUI app, it was often easiest to see errors in the code via our frontend. We could then update the backend and/or frontend to fix the bugs that we'd found. The one piece of integration testing that was left until near the end of our project was integrating our CPU (Single Player mode) into our app. In order for this to happen, several developers familiar with the app and CPU scheduled a meeting to work through this challenge together.

Having automatic builds and automatically running our test suites when code is pushed to master would improve the quality of our integration testing.

## Validation Testing

As we were preparing our Week 12 presentation, we examined each functional and non-functional requirement that we listed in our SSD and checked that we completed them. We are proud of the fact that met 100% of our functional requirements!

## System Testing

This step would ideally be done via user testing so that we could have the users launch the app on their on OS and play BattleShip. This would be one of our next steps!

## Next Steps

If we were able to work on our project past the end of CISC/CMPE 320, we would love to add CI/CD. Incorporating writing build/test/deployment pipelines and conducting more rigorous code reviews would all be priorities in the next phase of development. User testing would also be important in improving the user experience.

## Specific Coding Assignments/Detailed Timeline

### Week 9: Integration of Frontend and backend

- ☒ Integration of ship placement screen (Eric Leuty Jeremy Browne)
- ☒ Integration of shot screen\* (Jeremy Browne Eric Leuty)
- ☒ Integration of options screen (Danielle Mott Isabelle Quail)
- ☐ Clean up of backend (Douglas Gowing Drew Anderson)
  - ☐ Change position attribute to be struct of <row, col, health>
    - Health should be changed to a boolean instead of an int
  - ☐ String Ship::"name" should be changed to "type"
  - ☐ In non-default constructor Ship constructor, initialize position vector of correct size
  - ☐ change Grid::writeShip() to private
  - ☐ in Grid::shoot, use loop to prompt user for valid shot instead of automatically switchings turns
    - Loop iterates based on shot return value

- ☐ Change name of shotsGrid to gridState
- ☐ Rename addShip to checkPosition and have it return a bool
- ☒ Finish and connect bot with our "normal" backend (Douglas Gowing Drew Anderson)

## Week 10

- ☒ Finish custom ship [Danielle Mott](#)
- ☒ Error checking [Danielle Mott](#)
- ☒ Any outstanding Week 9 backlog tasks (Isabelle Quail [Danielle Mott](#) Drew Anderson Eric Leuty Douglas Gowing Jeremy Browne)
- ☒ Add bus/bike/car UI ([Danielle Mott](#) Isabelle Quail)
- ☒ Add bus/bike/car UI into grid ([Danielle Mott](#) Isabelle Quail)
  - ☒ Inkscape
- ☒ Ensure that page navigation makes sense (when does it switch windows) (Jeremy Browne Eric Leuty)
  - ☒ Start game
  - ☒ Turn changes
  - ☒ Game over
- ☒ Integrate Bot with frontend (single player mode) (Douglas Gowing Drew Anderson)
- ☒ Work on shot UI (Drew Anderson Eric Leuty)
- ☒ Goal: app should be ready to be tested

## Week 11

- ☒ Set up testing framework ([Danielle Mott](#))
- ☒ Meeting to determine where we're at and clean up the UI (Isabelle Quail [Danielle Mott](#) Drew Anderson Eric Leuty Douglas Gowing Jeremy Browne)
  - ☒ Make it look nicer 😊
  - ☒ Not functionality related

## Week 12

- ☐ Advertisement poster ([Isabelle Quail](#) [Danielle Mott](#) [Drew Anderson](#) [Eric Leuty](#) [Douglas Gowing](#) [Jeremy Browne](#))
- ☒ Have completed code (Isabelle Quail [Danielle Mott](#) Drew Anderson Eric Leuty Douglas Gowing Jeremy Browne)
- ☒ Film video/presentation (Isabelle Quail [Danielle Mott](#) Drew Anderson Eric Leuty Douglas Gowing Jeremy Browne)