RAISING DEFORESTATION AWARENESS THROUGH ONLINE EDUCATION

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science
University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree
Bachelor of Science, School of Engineering

Robert Wallace
Spring, 2020

Technical Project Team Members
Trevor Bedsaul
Henry Clabby
Ryan Coulter
Sammy Hecht
Dylan Peters
Teddy Vallar
Robert Wallace

On my honor as a University Student, I have neither given nor received unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Signature Robert Wallace _____ Date 4/5/2020
Robert Wallace

Approved Ahmed S Ibrahim _____ Date 4/27/2020
Dr. Ahmed Ibrahim, Department of Computer Science

Table of Contents

Abstract

Our capstone team worked alongside the Amazon Aid Foundation to solve fundamental navigation and accessibility issues affecting the foundation's web-based, educational game called the "Grow A Tree" game. Our capstone team was motivated by the core mission of the Amazon Aid Foundation: to educate and spread awareness about the importance of the Amazon Rainforest, one of the planet's most vital natural assets in the fight against climate change which is being rapidly deforested by human destruction and uncontained forest fires. By solving the game's issues, our team aimed to answer the question: How can we increase the instructional value of the game and facilitate wide-spread use of the learning module across classrooms of North America, and hopefully the globe? This goal was accomplished by first establishing a working relationship with communications specialist for the Amazon Aid Foundation, Ben Eppard, to elicit and discuss the functional requirements that would guide our development process. We employed the Scrum methodology over the course of two semesters, aided by continuous integration (GitHub) and automated testing (TravicCI) tools. Our team was able to successfully complete all of the main requirements, resulting in a product that is ready for deployment and use by the public.

The development process, as a whole, taught us the importance of maintaining effective communication channels between team members and our client in order to effectively partition workloads, respond to changing product requirements, and overcome design roadblocks. In addition, because our work was a continuation of a prior capstone team's, we learned how important it is to restructure and refactor an existing codebase during the initial development

phases, adding documentation and comments where necessary, in order to streamline future development. Our team's work is significant because tropical rainforests often exist in countries that are rife with political corruption, as is the case with Brazil's Amazon. Our success in making the "Grow A Tree" game more usable and accessible therefore increases the amount of students that can be reached. These students who may go on to be the next generation's industry leaders, engineers, policy-makers, or the citizens who influence them, ultimately resulting in the greatest potential for a long term shift to good-governance of Amazonian resources.

List of Figures

1. Introduction

The Amazon Rainforest is one of our planet's most valuable defenses against climate change. Acting as the world's largest carbon sink, the Amazon absorbs around 600 million tons of carbon dioxide from our atmosphere every year. Unfortunately, human-related deforestation of tropical rainforests has been accelerating at an alarming rate across the globe. As noted by Lovejoy and Nobre (2018), two experts on matters of global biodiversity, the Amazon Rainforest comprises a majority of the planet's tropical forest, yet around 17% of this species-rich biome has already been destroyed as of 2018. They suggest that if deforestation climbs higher than 25% an ecological tipping point will be reached, shifting southern and central Amazonia from a tropical rainforest into a non-forest ecosystem called a degraded savannah (p. 1).

1.1 Problem Statement

The Amazon Aid Foundation, a Charlottesville non-profit, was established in 2010 with the core mission of educating global citizens on the importance of the Amazon Rainforest and the implications of its destruction; the foundation garners support to protect and restore the rainforest using artwork, film, and other multimedia projects. More specifically, Amazon Aid aims to put pressure on politically and economically powerful organizations because many continue to operate without any environmental accountability. E. Pereira, Ferreira, Ribeiro, Carvalho, and H. Pereira, experts in resource conservation and computational modeling, recorded the series of anti-environmentalist policies enacted by Brazilian politicians linked to the country's agribusiness since 2016. President Michel Temer eliminated multiple construction licenses previously required for companies cutting down the rainforest, and enacted other

policies reducing the public's ability to oversee those construction projects. After deforestation rates increased in 2016, Temer's administration cut the Brazilian Ministry of Environment's budget in half, then froze the budget at that level for a twenty-year period beginning in 2018. His successor Jair Bolsonaro promised to continue increasing access to the Amazon's resources (p. 8, 2019).

In August of 2017 The Amazon Aid Foundation began working with a prior UVa capstone team over the course of one academic year to develop a learning module that would educate and inspire middle school students on the importance of the Amazon. This educational, web-based game named "Grow A Tree" represented a long-term solution that could begin educating the next generation of environmentalists to boycott politicians and companies who choose to ignore the deforestation problem. Although this past UVa capstone team was successful in developing the core functionality of this application, the Amazon Aid Foundation was left with a new set of problems concerning the learning module. As with any grassroots effort, Amazon Aid was now concerned with expanding the learning module's outreach by getting the game into as many middle-school classrooms as possible, but fundamental issues with navigation and accessibility hindered its widespread use across public schools in North America and globally. At the time overall site navigation was minimal and users had no way of returning to previously completed levels of the game making class-wide discussions difficult. In addition, the learning module required password-protected accounts to track progress between logins. This requirement posed a serious barrier to entry because public middle school teachers require special administrative permission to use any website that requires students to create standalone

accounts. It became the current capstone team's goal to solve these issues for the Amazon Aid Foundation.

1.2 Contributions

Our capstone team was first able to successfully solve the navigational issues of the game. We redesigned the game's views to allow for intuitive backward and forward navigation throughout all levels of the game for both students and teachers, as well as fixing bugs related to transitions after a level or task is completed. Our team was also able to reduce barriers to entry to make the learning module viable for use in public schools. We accomplished this by restructuring the login process to use a class-wide code, distributed by the teacher, followed by a personalized four-digit code so that students can track their progress without needing to create an account with an email and password. In addition, we added teacher administration functionality to the learning module that allows teachers to track individual student progress and maintains information on student progress between class sessions.

2. Related Work

With the Amazon Rainforest being the world's largest tropical rainforest and overall environmental concerns an issue for the majority of the United States and the world (Figure 1, page 10), Amazon Aid is not alone in their mission. Two other online learning outlets, Rainforest Alliance and ARCAmazon, have similar missions and purposes when it comes to improving environmental responsibility in the world.

Rainforest Alliance is "an international non-profit organization working at the intersection of business, agriculture, and forests to make responsible business the new normal." (Rainforest Alliance, 2020). They offer many games for children to play to learn more about rainforests around the globe, and raise millions of dollars to help improve the life of at risk rainforests. However, Amazon Aid is able to do a better job than Rainforest Alliance can in regards to the Amazon Rainforest itself. Amazon Aid focuses entirely on the Amazon Rainforest, which enables it to maximize children's education on the struggles of the region.

ARCAmazon focuses all its efforts on the Amazon region, aiming to "to offer sustainable livelihoods for local people through responsible ecotourism, sustainable rainforest products and services, agroforestry/analog forestry, reforestation, PES/REDD+, education programs, media awareness and technology innovation." (ARCAmazon, 2020) However, ARCAmazon currently does not have any online learning features aimed towards children, something that Amazon Aid specializes in.

By focusing on both youth education and specializing in the Amazon Rainforest, Amazon Aid is able to do the most good of any other organization that is out there for their mission of "fighting for the Amazon through arts, sciences, and education" (Amazon Aid, 2020).

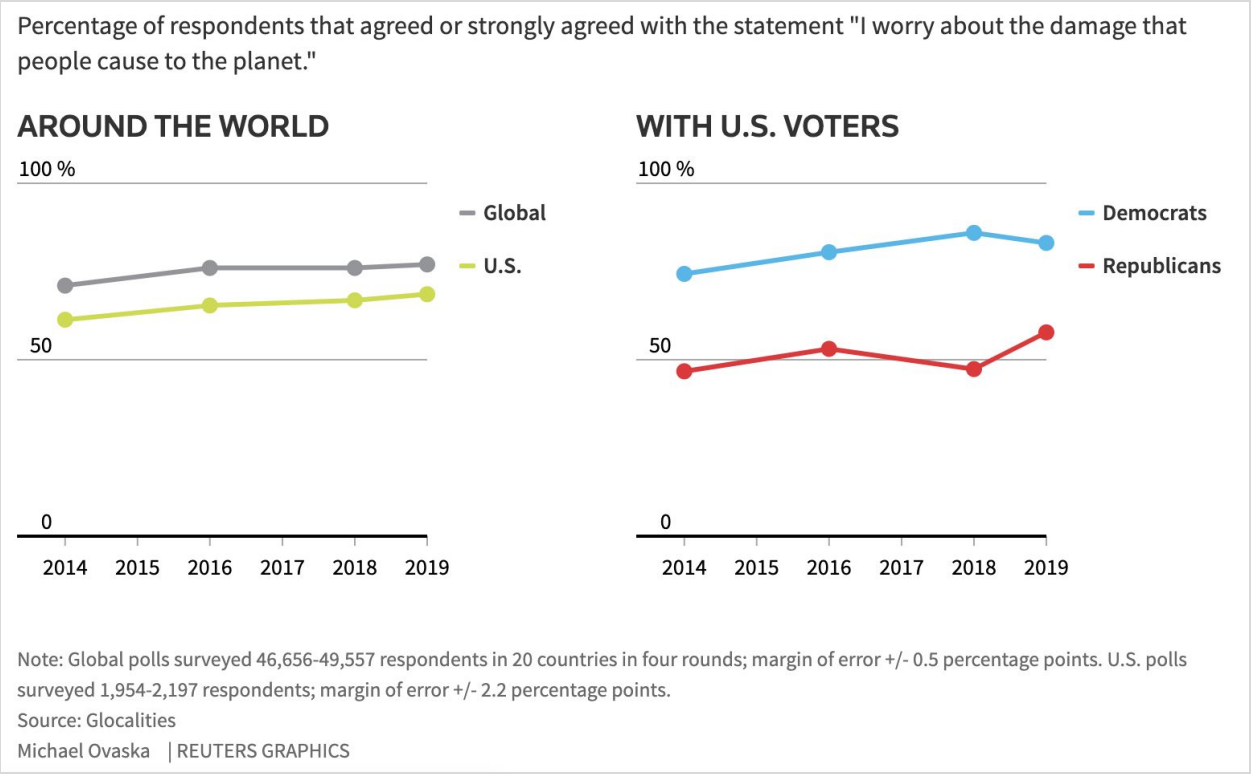

Figure 1: Rise in Environmental Concern: Concern about the environment is rising among both political parties in the United States, but slowly. (Reuters, 2019)

3. System Design

The AmazonAid foundation provides an online educational platform to educate users about the importance of the Amazon rainforest. Individual users have full access to the platform, and can register and sign in using an email and password. Teachers have the ability to register themselves and their students and are given a class access code upon sign up. With an individualized student access code generated by the teacher, students can then sign in and use the platform. Teachers also have the ability to track their student's progress and even reset their progress if needed. As this system is a continuation of a past project, we used the framework the project was created with, Django. The code for our project is under the MIT license.

3.1 System Requirements

System requirements are the basis for a project and the first thing that should be talked about between client and developer. System requirements encapsulate the operational desires of the client, placing these desires into a list of functional requirements stipulating what each participant, such as a user, admin, etc., should be able to do. It is through these requirements that the developers will design and build the system the client wants, at each step of development making sure each of the requirements are either implemented or can easily be implemented within the system design. Additionally, enumerating the requirements the client has for the project will also provide a lense through which the development timeline can be understood. Each of the system requirements will have development requirements varying in the amount of time each feature takes to implement and when in the process it can be implemented if it relies on other system features. Using this information, the developers can plan each sprint to be as

effective as possible and shrink development time. The requirements for our project with Amazon Aid can be found below:

MINIMUM REQUIREMENTS

- As a USER I should be able to go back to previously completed levels within each “tree” when using the online learning platform.
- As a USER I should be able to enter the learning platform through a single teacher login.
- As a USER I should be able to enter the learning platform using a teacher’s login information as well as a personal avatar, so that each student doesn’t have to create an account.

DESIRED REQUIREMENTS

- As an ADMIN I should be able to add images that are persisted in a database, so that content can be added and served reliably.
- As a USER I should be able to bypass the integrated minigames in order to move onto more content.

OPTIONAL REQUIREMENTS

- As a USER I should be able to access and play a variety of minigames which are native to the web app while progressing through learning content.

3.2 Wireframes

Wireframes are an essential part of early communication between client and developers. Unlike general functionality, aesthetics and design can be difficult to communicate and misunderstandings between client and developers could lead to expensive and time-consuming changes late in the design process. Wireframes are a way to mitigate this risk by allowing the

client and developer to agree on a general website design without any coding needed. By beginning the conversation on design before development starts, the development team can enter the first development phases with a template to use and a firm mutual ground to rely on when communicating with the client for the questions that arise. The wireframes for our project with Amazon Aid can be found below:

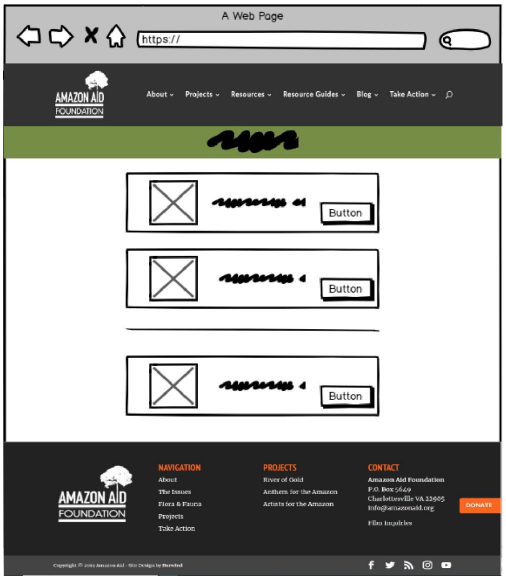

Figure 2: Choose Login: The wireframe shows users three options for ways to log in, accompanied by an explanatory icon and text.

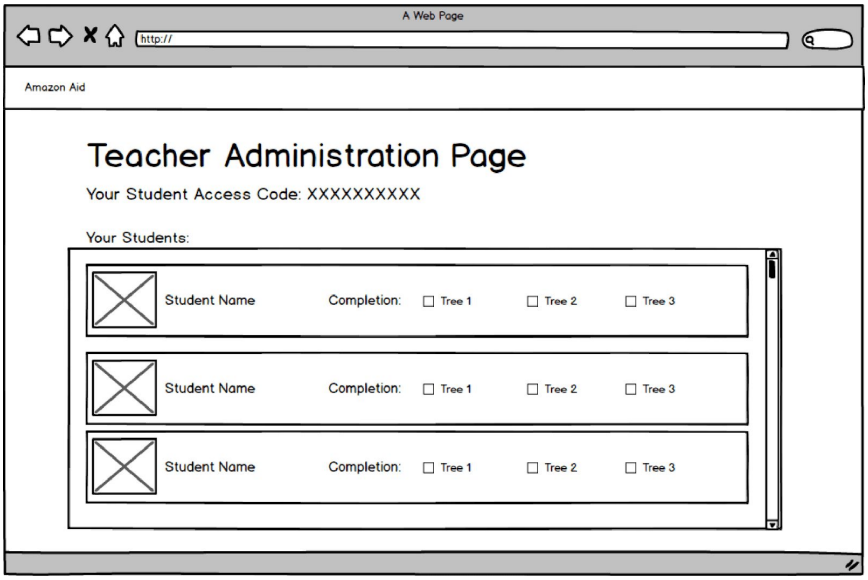

Figure 3: Teacher Admin: A wireframe shows the page where teacher users can view their students' progress through the game, with each student identified by an icon and name.

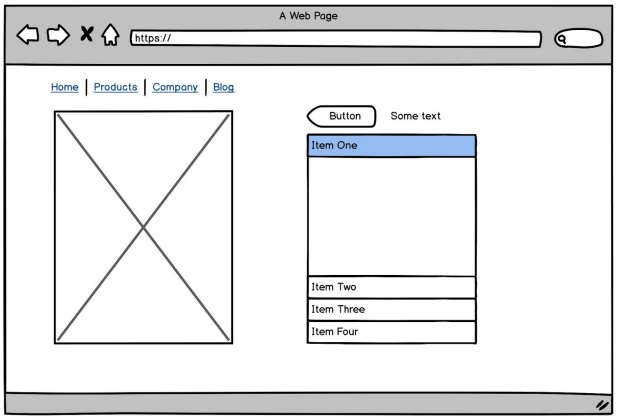

Figure 4: Tree Game: The tree games would show the appropriate tree picture on the left with each level's tasks on the right.

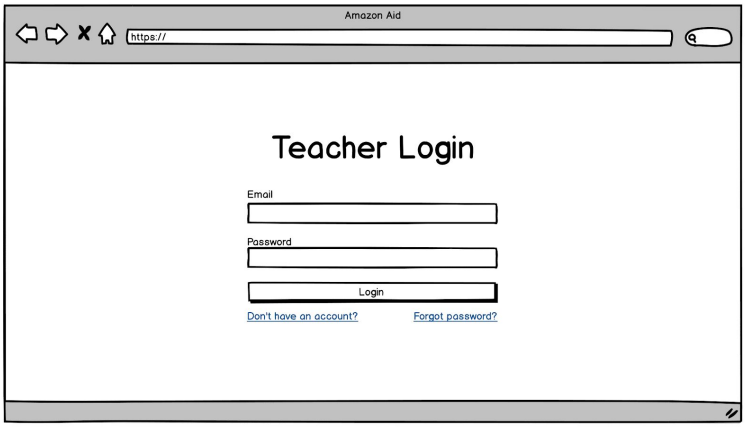

Figure 5: Teacher Login: Teacher users would log in with an email and password.

3.3 Sample Code

Model Functions:

User Model: Add Student

```
def add_student(self):
    if self.userType != 1: # don't run method if not a teacher
        return

    seed(15)

    flag = True
    all_avatars = [# string list of avatar file names, removed to save space]
    students = self.students # currently a string
    if students is None:

        # get random value for pin, don't have to check if it exists cause it is the first one
        value = random()
        new_pin = int(1000 + (value * (8999)))
        uname = str(self.accessCode) + "-" + str(new_pin)
```

```
            # register new student
            newStudent = User.objects.create_user(username = uname, activeTreeNum = 0,
accessCode=self.accessCode, userType=2)
            newStudent.save()

            # get avatar for the new student (it will also be the first avatar)
            image_index = 0
            image = all_avatars[image_index]

            # fill in the teacher's stuff
            response = [{"id": newStudent.id, "pin": new_pin, "img": image}] # have an image name for now
just to have something
            self.students = json.dumps(response) # dumps will submit it as a string
            self.save()
        else:

            #loop until we find a code not yet taken
            while(flag):
                flag = False
                # random float between 0 and 1
                value = random()

                # cast to int and make it between 1000 and 9999
                new_pin = int(1000 + (value * (8999)))

                # get student list, and loop to see if pin exists. Also need to do this for the images once we have that
                list_students = json.loads(students) # makes it a list
                for student in list_students:
                    pin = student['pin'] # get pin from student
                    # Are pins equal? if so, flag is true and we need a new random pin
                    if int(pin) == new_pin:
                        flag = True
                        break

            #we have now found a pin that isn't used, so register the student
            #print(str(self.accessCode) + "-" + str(new_pin))
            uname = str(self.accessCode) + "-" + str(new_pin)
```

```
        newStudent = User.objects.create_user(username = uname, activeTreeNum = 0,
accessCode=self.accessCode, userType=2)
        newStudent.save()

        ## get the next avatar in list but finding the current number of students, and getting the next image
index
        image_index = len(list_students) % len(all_avatars)
        image = all_avatars[image_index]
        # add the student to the teacher's json list of students
        response = {"id": newStudent.id, "pin": new_pin, "img": image} # this is a list element
        list_students.append(response)
        self.students = json.dumps(list_students) # dumps will submit it as a string
        self.save()
```

User Model: Get level completion

```
def get_level_completion(self):
    trees = TreeNew.objects.all()
    all_levels = []
    for tree in trees:
        level_num = tree.get_current_level(self).number
        if level_num==0:
            level_num = 1 # This is checking for the "intro" level in the first tree so that it doesn't print negative
        tot_levels = tree.get_num_levels()
        if level_num == tot_levels and tree.get_current_level(self).is_complete(self):
            all_levels.append([level_num, tot_levels])
        else:
            all_levels.append([level_num-1, tot_levels])
    return all_levels
```

Level Model: is_complete

```
def is_complete(self, user):
    complete_tasks = self.tasks.filter(completed_users=user)
    return len(complete_tasks) == len(self.tasks.all())
```

Views:

Teacher Admin:

```
@login_required(login_url="login")
def teacher_admin(request):
    if request.user.userType != 1:
        return HttpResponseRedirect(reverse('index'))
    user = request.user
    all_level_data = []

    if user.students != None:
        students = json.loads(user.students)
        for student in students:
            student_code = str(student['pin'])
            stud_user = User.objects.get(id=student['id']) #this only gets one student
            user_img = static(user.get_student_image(student['pin']))
            levels = stud_user.get_level_completion
            all_level_data.append([student_code, levels, user_img])
    context = {
        'data': all_level_data,
        'teacher_code': user.accessCode,
    }

    if request.method == 'POST':
        print(request.POST)
        user = request.user
        if 'add_student' in request.POST:
            # see method in models.py
            user.add_student()
        if 'reset_student' in request.POST:
            if user.students != None:
                students = json.loads(user.students)
                for student in students:
                    student_code = str(student['pin'])
                    stud_user = User.objects.get(id=student['id'])
                    stud_user.reset_user_progress()
        if 'reset_individual_student' in request.POST:
```

```
            # get the student code from the hidden input in the form
            student_code = request.POST.get('student_code')
            # build the student username
            stud_username = user.accessCode + "-" + str(student_code)
            stud_user = User.objects.get(username=stud_username)
            # reset the progress of the one individual student
            stud_user.reset_user_progress()
        # Download Info button
        elif 'download_info' in request.POST:
            # Create the HttpResponse object with the appropriate CSV header.
            response = HttpResponse(content_type='text/csv')
            response['Content-Disposition'] = 'attachment; filename="studentProgress.csv"'
            writer = csv.writer(response)

            # write header row of the csv
            writer.writerow(['Student Code', 'Tree 1', 'Tree 2', 'Tree 3'])
            if user.students != None:
                students = json.loads(user.students)
                # for each student in the class
                for student in students:
                    stud_user = User.objects.get(id=student['id']) # this returns a student
                    levelProgress = stud_user.get_level_completion() # returns a list of pairs
                    writer.writerow([str(student['pin']), levelProgress[0][0], levelProgress[1][0], levelProgress[2][0]])

            return response

    return render(request, 'teacher_admin.html', context)
```

Login Student View:

```
def login_student_view(request):
    form = LoginStudentForm()
    alerts = []
    alert = False
    if request.method == 'POST':
        form = LoginStudentForm(request.POST)
        if form.is_valid():
```

```
        # get both access codes from the form
        teach_access_code = form.cleaned_data['teacher_access_code']
        student_access_code = form.cleaned_data['student_access_code']



        # teacher access code does not exist
        if not User.objects.filter(accessCode=teach_access_code).exists():
            alerts.append('Teacher access code is incorrect')
            alert = True
        else: # teacher access code does exist
            teacher = User.objects.get(accessCode=teach_access_code, userType=1)
            curr_student = None
            students = json.loads(teacher.students)

            # find the student
            for student in students:
                for key, val in student.items():
                    if key == "pin" and val == student_access_code:
                        curr_student = student

            # student exists
            if not curr_student == None:
                # authenticate the student and login
                student_user = authenticate(username=str(teach_access_code) + "-" + str(student_access_code))
                login(user=student_user, request=request,
backend="policypiece.student_auth.PasswordlessAuthBackend")
                context = {'user': student_user}
                return HttpResponseRedirect(reverse('index'), context)
            else: # student doesn't exist
                alerts.append('Could not find student account')
                alert = True
    else:
        alerts.append('form invalid')
        alert = True
  return render(request, 'login_student.html', {'form': form, 'alerts': alerts, 'alert': alert})
```

Add Article for Game:

```
@login_required(login_url="login")
def add_article(request):
    if request.user.isAdmin == 0:
        return HttpResponseRedirect(reverse('index'))

    if request.method == "POST":
        form = ArticleForm(request.POST)
        if form.is_valid():
            # Save article
            title = form.cleaned_data['title']
            category = form.cleaned_data['category']
            _base64content = form.cleaned_data['_base64content']
            if LearnContent.objects.filter(title=title).exists():
                form.add_error('title', 'Must be unique.')
            else:
                article = LearnContent.objects.create(
                    title=title,
                    category=category,
                    _base64content=_base64content
                )
                article.save()

                # Save associated task if level was selected
                level_id = int(form.cleaned_data.get('level_id'))
                if level_id != 0:
                    level = Level.objects.get(pk=level_id)
                    task = Task.objects.create(
                        name='Article Task',
                        description='See article',
                        level=level,
                        article=article
                    )
                    task.save()
```

```
        # enable this page if the user wants to redirect automatically after
        # adding a new article, else page will refresh to new add form
        # return redirect('view_articles')
    else:
        form = ArticleForm()

    trees = TreeNew.objects.all().order_by('pk')
    levels_by_tree = {}
    for tree in trees:
        levels = tree.get_ordered_levels()
        levels_by_tree[tree.id] = json.loads(serializers.serialize('json', levels))
    levels_by_tree = json.dumps(levels_by_tree)

    context = {
        'form': form,
        'trees': trees,
        'levels_by_tree': levels_by_tree
    }
    return render(request, 'admin_dashboard/articles/addarticle.html', context)
```

Forms:

Login Student Form:

```
class LoginStudentForm(forms.Form):
    teacher_access_code = forms.CharField(label='Teacher Code', max_length=10)
    student_access_code = forms.IntegerField(label='Student Code')
```

Article Form (articles for game):

```
class ArticleForm(forms.Form):
    CATEGORIES = (
        ('Article', 'Article'),
        ('Game', 'Game'),
        ('Video', 'Video')
    )
```

```
    title = forms.CharField(label='Title', max_length=100)
    category = forms.ChoiceField(label='Category', widget=forms.Select, choices=CATEGORIES)
    _base64content = forms.CharField(max_length=20000, widget=forms.Textarea, required=False)
    # Optional level info
    level_id = forms.CharField(label='Level', required=False)
```

Create Post Form:

```
class CreatePostForm(forms.Form):
    title = forms.CharField(label='Title', max_length=100, required=True)
    description = forms.CharField(label='Description', widget=forms.Textarea, required=True)
    tags = forms.CharField(label='Tags', max_length=100, required=False)
```

3.4 Sample Tests

Testing is one of the most important components in the software development lifecycle. It is a never ending process that helps ensure that the system is built optimally. At its core, it is used to find defects in an application by comparing the actual result an application produces against the result that is expected. Doing so verifies that the product conforms to whatever requirements that have been set for it, which is especially important when making an application for another group or company.

Many tests we created dealt with the new user types we created. For instance the two tests shown below test to make sure the permissions for teachers and students are correct. The first test ensures a teacher can change their display name correctly, while the second ensures that a student is not able to change their display name as they have reduced permissions.

```
def test_change_displayName(self):
    '''
   Checks to see if a user's DisplayName has changed.
   '''
    original_displayName = 'original'
    user = models.User.objects.create_user(username='Person5@gmail.com', email='Person5@gmail.com',
                                           first_name='Poster',
                                           last_name='McGee', displayName=original_displayName, password='hoops',
                                           country="United States", city="Fredericksburg", isAdmin=False,
                                           private=False)
    self.client.login(email="Person5@gmail.com", password="hoops")
    form_data = {'email': 'Person5@gmail.com', 'password': 'hoops'}
    response = self.client.post(reverse('login'), form_data, follow=True)

    form = {'displayName': 'PepeHead'}
    response2 = self.client.post(reverse('changesettings'), form, follow=True)
    user = models.User.objects.get(email="Person5@gmail.com")
    new_displayName = user.displayName
    self.assertEqual(original_displayName, new_displayName)
```

```
def test_invalid_change_displayName_student(self):
    '''
   Checks to see if a user's DisplayName has changed.
   In this case a student tries to change it, so the
   change should not go through and the before and after names
   should be equal
   '''
    original_displayName = 'before'
    user = models.User.objects.create_user(username='Person5@gmail.com', email='Person5@gmail.com',
                                           first_name='Poster',
                                           last_name='McGee', displayName=original_displayName, password='hoops',
                                           country="United States", city="Fredericksburg", isAdmin=False,
                                           private=False, userType=2)
    self.client.login(email="Person5@gmail.com", password="hoops")
    form_data = {'email': 'Person5@gmail.com', 'password': 'hoops'}
    response = self.client.post(reverse('login'), form_data, follow=True)

    form = {'displayName': 'after'}
    response2 = self.client.post(reverse('changesettings'), form, follow=True)
    user = models.User.objects.get(email="Person5@gmail.com")
    new_displayName = user.displayName
    self.assertEqual(original_displayName, new_displayName)
```

The three tests below also deal with the new user type system. In each test, one of the three user types is created and is then tested to see if the corresponding user type is correct.

```python
# Test correct propagation of the user model as an individual
def test_user_model_individual(self):
    user = models.User.objects.create_user(username='Person5@gmail.com', email='Person5@gmail.com',
                                           first_name='Poster',
                                           last_name='McGee', password='Hoops123', country="United States",
                                           city="Fredericksburg", isAdmin=False,
                                           private=False, userType=0)
    self.assertEqual(user.userType, 0)


# Test correct propagation of the user model as a teacher
def test_user_model_teacher(self):
    user = models.User.objects.create_user(username='Person5@gmail.com', email='Person5@gmail.com',
                                           first_name='Poster',
                                           last_name='McGee', password='Hoops123', country="United States",
                                           city="Fredericksburg", isAdmin=False,
                                           private=False, userType=1)
    self.assertEqual(user.userType, 1)

# Test correct propagation of the user model as a student
def test_user_model_student(self):
    user = models.User.objects.create_user(username='Person5@gmail.com', email='Person5@gmail.com',
                                           first_name='Poster',
                                           last_name='McGee', password='Hoops123', country="United States",
                                           city="Fredericksburg", isAdmin=False,
                                           private=False, userType=2)
    self.assertEqual(user.userType, 2)
```

Other tests that were done focus more on the behavior of the code base rather than the behaviors that potential users could exhibit. The two tests shown below make sure the class access code generated for a teacher meets the two specified requirements of being an alphanumeric code that is eight characters long.

```python
def test_access_code_length(self):
    '''
    Ensure the length of the teacher access code is 8
    '''
    form_data = {}
    form_data['fName'] = 'Gerald'
    form_data['lName'] = 'Chad'
    form_data['email'] = 'gerald@gmail.com'
    form_data['private'] = False
    form_data['displayName'] = 'Gerald'
    form_data['password'] = 'passWord3'
    form_data['confirmPassword'] = 'passWord3'
    form_data['city'] = ''
    form_data['isAdmin'] = False
    form_data['country'] = 'United States'
    form_data['subscribed'] = False

    # submit the form
    response = self.client.post(reverse('registerteacher'), form_data, follow=True)

    # see if user is logged in
    user = response.context['user']
    self.assertEqual(len(user.accessCode), 8)
```

```
def test_access_code_content(self):      rjcoulter, 3 months ago • rjc8ez sprint 6 unit tests
    '''
    Ensure the access code only contains alphanumeric characters
    '''
    form_data = {}
    form_data['fName'] = 'Gerald'
    form_data['lName'] = 'Chad'
    form_data['email'] = 'gerald@gmail.com'
    form_data['private'] = False
    form_data['displayName'] = 'Gerald'
    form_data['password'] = 'passWord3'
    form_data['confirmPassword'] = 'passWord3'
    form_data['city'] = ''
    form_data['isAdmin'] = False
    form_data['country'] = 'United States'
    form_data['subscribed'] = False

    # submit the form
    response = self.client.post(reverse('registerteacher'), form_data, follow=True)

    # see if user is logged in
    user = response.context['user']
    self.assertTrue(user.accessCode.isalnum())
```

3.5 Code Coverage

Code coverage will be tracked using the coverage.py python package alongside django built in testing. The coverage.py package can be installed on linux systems using the following command:

sudo apt-get install python-coverage

To use coverage.py to track code coverage in testing a django system, navigate to the folder containing manage.py as you would for running django tests normally but run:

coverage run --source='.' manage.py test

Run the following commands to 1) generate the results in the terminal or 2) generate a navigable html doc to see what lines ran and which didn't:

1) coverage report
2) coverage html

Code coverage results

File	Total Line	Number Missed	Percent Covered
AmazAid__init__.py	0	0	100%
AmazAid\secret.py	19	0	100%
AmazAid\settings.py	66	0	100%
AmazAid\urls.py	8	0	100%
AmazAid\wsgi.py	4	4	0%
manage.py	7	0	100%
policypiece__init__.py	0	0	100%
policypiece\admin.py	28	0	100%
policypiece\apps.py	3	3	0%
policypiece\forms.py	87	0	100%
policypiece\migrations\0001_initial.py	11	0	100%
policypiece\migrations\0002_auto_20191027_2005.py	6	0	100%
policypiece\migrations\0002_auto_20191105_2028.py	5	0	100%
policypiece\migrations\0003_auto_20191027_2006.py	5	0	100%
policypiece\migrations\0003_auto_20191106_1555.py	6	0	100%
policypiece\migrations\0004_merge_20191111_1547.py	5	0	100%
policypiece\migrations\0005_auto_20191114_1257.py	6	0	100%
policypiece\migrations\0006_auto_20191114_1318.py	5	0	100%
policypiece\migrations\0007_auto_20191114_1853.py	6	0	100%
policypiece\migrations\0007_auto_20191118_1134.py	5	0	100%
policypiece\migrations\0007_auto_20191120_1946 2.py	5	0	100%
policypiece\migrations\0007_auto_20191120_1946.py	5	0	100%
policypiece\migrations\0008_auto_20191118_1131.py	6	0	100%
policypiece\migrations\0008_auto_20191120_2018.py	5	0	100%
policypiece\migrations\0009_merge_20191125_1313 2.py	5	0	100%
policypiece\migrations\0009_merge_20191125_1313.py	5	0	100%
policypiece\migrations\0010_merge_20191201_1704.py	5	0	100%
policypiece\migrations\0011_auto_20191202_1133.py	5	0	100%
policypiece\migrations\0011_auto_20191203_1945.py	6	0	100%
policypiece\migrations\0011_auto_20191206_1108.py	5	0	100%
policypiece\migrations\0011_merge_20191204_1052.py	5	0	100%

policypiece\migrations\0012_auto_20191209_1607.py	5	0	100%
policypiece\migrations\0013_auto_20191209_1607.py	6	0	100%
policypiece\migrations\0014_auto_20191209_1613.py	5	0	100%
policypiece\migrations\0015_auto_20191209_1630.py	6	0	100%
policypiece\migrations\0016_auto_20191209_1646.py	6	0	100%
policypiece\migrations\0017_auto_20191212_1350.py	5	0	100%
policypiece\migrations\0017_merge_20191212_1112.py	5	0	100%
policypiece\migrations\0018_merge_20200128_0849.py	5	0	100%
policypiece\migrations\0018_merge_20200128_1048.py	5	0	100%
policypiece\migrations\0019_merge_20200128_1202.py	5	0	100%
policypiece\migrations\0020_merge_20200128_2130.py	5	0	100%
policypiece\migrations\0021_user_studentcode.py	5	0	100%
policypiece\migrations\0022_auto_20200211_1916.py	5	0	100%
policypiece\migrations\0023_auto_20200211_1932.py	5	0	100%
policypiece\migrations__init__.py	0	0	100%
policypiece\models.py	288	24	92%
policypiece\pipeline_data.py	2	0	100%
policypiece\processors.py	7	0	100%
policypiece\serializers.py	29	0	100%
policypiece\student_auth.py	14	4	71%
policypiece\templatetags__init__.py	0	0	100%
policypiece\templatetags\active_tag.py	11	1	91%
policypiece\templatetags\json.py	13	7	46%
policypiece\templatetags\split_tag.py	4	0	100%
policypiece\tests.py	2821	8	99%
policypiece\urls.py	4	0	100%
policypiece\views__init__.py	11	0	100%
policypiece\views\changesettingsview.py	79	0	100%
policypiece\views\contactview.py	19	0	100%
policypiece\views\frontpageview.py	42	0	100%
policypiece\views\loginstudentview.py	44	4	91%
policypiece\views\loginview.py	20	0	100%
policypiece\views\logoutview.py	7	0	100%
policypiece\views\manageusersview.py	39	0	100%

policypiece\views\registerteacherview.py	64	8	88%
policypiece\views\registerview.py	55	0	100%
policypiece\views\remainingviews.py	1015	216	79%
policypiece\views\voiceview.py	17	0	100%
TOTAL	5017	279	94%

3.6 Installation Instructions

3.6.1 Initial AWS Account Setup

- First, create an AWS account at aws.amazon.com(aws.amazon.com) and follow their instructions
- Once done, login to AWS

3.6.2 Create an EC2 instance and download AWS private key

- Click "Services" at top left, click "EC2" from drop-down
- On the left side, under the "Instances" heading click "Instances"
- Click "Launch Instances"
- Select the first option "Amazon Linux AMI 2018.03.0 (HVM), SSD Volume Type" by clicking "Select" to the right.
- Click "Review and Launch"
- Click "Launch"
- From the drop-down menu, select "Create a new key pair" and enter in a key pair name. Click "Download Key Pair". Once downloaded, proceed by selecting "Launch Instances"
- Go back to the instances page (Services –> EC2 –> Instances)

- Click on the newly created instance and keep track of the URL (Ex: ec2-184-72-111-17.compute-1.amazonaws.com)

By now, you should have an EC2 instance and an AWS private key. **IMPORTANT**: You **MUST** wait for the **Status Checks** column to give you a green arrow for the instance before you can connect to it. It will say "2/2 checks..." when done.

3.6.3 Setup security group for EC2

- Go to Services –> EC2 –> Instances
- Click the instance and select the associated security group, e.g. “launch-wizard-x” etc.
- Click on the security group and click on Actions –> Edit Inbound Rules
- Under Type select "All traffic"
- Under Source select "Anywhere"
- Click "Save"

As of now, you are finished with the AWS console.

3.6.4 Edit permissions of AWS private key

- Open up a new terminal/console window.
- Navigate using cd to where the AWS private key was downloaded, such as Downloads by running “cd Downloads” in terminal.
- Once there, run the following command:

sudo chmod 400 whateveryourprivatekeyisnamed.pem

3.6.5 SSH to AWS server

- Now that you have the private key and the EC2 URL, in the terminal, run the command: ssh -i /path/to/private/key.pem ec2-user@(EC2 URL)
- For example: ssh -i ~/Downloads/TestServer.pem ec2-user@ec2-184-72-111-17.compute-1.amazonaws.com
- set the current domain into an environment variable using the EC2 instance's public IP: export CURRENT_DOMAIN='http://EC2-IP-ADDRESS:8000/'. Don't forget the http:// and the ending slash/.

3.6.6 Create S3 Bucket

- To create the S3 bucket, go back to the AWS console and navigate to the S3 service.
- Create the bucket and turn off the block on public access during configuration
- Once the bucket is created, click your name in the top corner and select "My Security Credentials" from the drop down.
- On the Security credentials page, click the Access Key tab and create an access key for the S3 service.
- Save the information presented because it will be used during the configuration of the secret.py file

3.6.7 Install Project and Dependencies

- Once you have connected to the EC2 server, run the following commands:

sudo yum install git-core

git clone https://github.com/uva-cp-1920/AmazonAid.git

cd AmazonAid/src/amazonaid/

sudo yum install mysql-server

sudo yum install gcc

sudo yum install mysql-devel

sudo pip install -r requirements.txt

3.6.8 Create the secret.py file in ~/AmazonAid/src/amazonaid/AmazAid/AmazAid/

- Obtain the **secret.py** file and the **MySQL database password**. The **secret.py** file template can be obtained from the github repo where the files were downloaded from in the ./AmazAid/AmazAid/ directory.
- This "secret.py" file will be placed in ~/AmazonAid/src/amazonaid/AmazAid/AmazAid/
- The secret.py file will contain all information about the database that is running locally on the EC2 server, as well as the information needed to perform GETs from the S3 bucket where all website assets are stored.
- Fill in the information in the template file, either on the EC2 or locally.
- The database setting should be: database_engine = "django.db.backends.mysql"
- The database user should be: database_user = "root"
- The S3 information should be the key info obtained during the S3 bucket creation step and the name of the bucket in S3.

- If copying secret.py from a local machine to AWS, locally run the command: scp -i ~/path/to/key/file.pem file1 user@ec2_elastic_ip:/home/ec2-user/AmazonAid/src/amazonaid/AmazAid/AmazAid/
- Elastic IP can be obtained by using the following instructions:
- Open the Amazon EC2 console at https://console.aws.amazon.com/ec2/.
- In the navigation pane, choose Elastic IPs.
- Choose Allocate new address.
- For IPv4 address pool, choose Amazon pool.
- Choose Allocate, and close the confirmation screen.

3.6.9 Login to MySQL and set up database

* The databasepassword below corresponds to the MySQL database password given above

```
sudo service mysqld start
mysql -u root
create database slp_amazonaid;
grant all on slp_amazonaid.* to 'amazonaid' identified by 'databasepassword';
grant all on slp_amazonaid.* to 'amazonaid'@'localhost' identified by 'databasepassword';
exit
```

3.6.10 Collect static files and migrate the database

- Back in the terminal, run the following commands:

```
mkdir ~/AmazonAid/src/amazonaid/AmazAid/policypiece/static/
cd ~/AmazonAid/src/amazonaid/
```

python AmazAid/manage.py collectstatic

python AmazAid/manage.py makemigrations

python AmazAid/manage.py migrate

Note: collectstatic may take a while

3.6.11 Webservice Setup

- In the terminal, run the following commands:

sudo yum install nginx
sudo pip install gunicorn

sudo service nginx start

sudo yum install python36

sudo yum install python36-devel python36-pip

sudo yum install python36-setuptools python36-virtualenv

sudo python3 -m pip install gunicorn

cd ~/AmazonAid/src/amazonaid/

sudo python3 -m pip install -r requirements.txt

cd ~/AmazonAid/src/amazonaid/AmazAid/

nohup gunicorn -b (EC2 URL):8000 AmazAid.wsgi:application &

The website should now be at (EC2 URL):8000

For example, ec2-184-72-111-17.compute-1.amazonaws.com:8000

4. Results

Our system was successful in satisfying nearly all of the requirements set forth by the customer. However, we have not yet integrated native minigames, and we have not created an intuitive system for the administrator to add content. The system now includes login options for individuals, teachers, and students. Teachers have the ability to monitor and administer students via a teacher administrator view. Additionally, students are able to be identified by unique avatars, and easily navigate the game levels through previous and next buttons. Students are also now able to bypass minigames through a password distributed to teachers. Finally, students can now post their voices about the Amazon directly within the game rather than having to navigate to the voices view.

One of the largest requirements for this project was a rework of the login system. In order to protect student's privacy, we had to come up with a login system that didn't require them to enter any important information about themselves. The user model was modified to include an account type identifier ranging from zero to two. These identifiers map to individual, teacher, and student accounts. When a teacher registers an account, 46 unique student accounts are auto-registered and assigned a student access code. Students can then login to the system using the teacher's class access code in addition to their unique student access code. This system allows students to login without disclosing any personal information, and it allows teachers to monitor student progress via a teacher admin page. Teachers can assign and identify students by their unique avatars, and teachers can reset the progress of students on the teacher admin page.

Finally, the teacher admin page allows the teacher to download a csv file to easily work with the student data. This new system protects the confidentiality of student information, and gives teachers the ability to monitor and control student accounts.

Our system also made changes to improve the flow and ease of use of the game. One of these changes was implementing forward and back buttons to allow users to easily visit different levels of the game. Before, users could only progress through the game sequentially, but with this change they are able to explore all of the content at whatever time they wish, up to the latest level they've unlocked. In addition to these buttons, a change was made to allow students to post their views on the Amazon directly within the game. The previous system required students to visit the voices page to post their thoughts, but the new system integrates this functionality directly into the game. These changes made the game more intuitive and packaged more functionality directly into the game.

The customer uses this system as an administrator. The customer can modify content by using the Django admin page, and by uploading static assets to the S3 bucket. The customer can also monitor and approve posts made by users to ensure that they are appropriate. Aside from modifying content and approving posts, the customer only needs to provide teachers and individuals with a link to the web application in order to put the product into action.

Users interact with this system according to their needs and user type. Individuals may create an account and progress through the game or post their voices as they wish. Teachers, however, use the system to distribute accounts to students and monitor progress. The teacher admin page provides teachers with all of the student account information, and allows them to see and reset progress. Teachers may wish to download this information in order to easily distribute

student logins or work with student data in csv format. Finally, students use the game much like individuals, but their login process is much different. Students are given a class access code and a student access code by their teacher. These access codes allow students to login without ceding any personal information, and they allow students to be associated with their teacher for progress monitoring. Students are identified by unique avatars, and their progress is tracked on the teacher admin page as they progress through the game. Prior to the rework of the login system students would be required to enter six pieces of personal information, but the new system allows students to access the system without disclosing any information about themselves.

In conclusion, the work on this system both improved the login process, and made the game more user friendly. In order to post a voice, users now save three extra clicks, and do not need to interrupt their progress. The login system now includes three distinct user types in order to directly cater to the needs of different users. Teachers are now automatically given 46 prepopulated student accounts with unique avatars. Student accounts now require zero pieces of personal information as opposed to the six needed in the previous system. In addition to these improvements, all users are now able to progress forwards and backwards in the game in order to interact with the content however they choose to. These changes allow the system to be deployed in schools, and make the game much more friendly to any user that chooses to engage with it.

5. Conclusions

The Amazon Aid Foundation returned to UVa to work with another capstone team because despite the merit of their learning materials, minor obstacles to adoption for teachers and difficulties navigating for students prevented the site from spreading through middle schools. In analyzing the problem, the team reevaluated the relative importance of different stakeholders involved. While the site's educational value derives from the experiences of students and individuals, teachers are the most important actors in the site's adoption and diffusion process. Details of the login process requiring teachers to request special administration to use the site could determine whether they share the Foundation's message with twenty or more students per class. When using the site, if teachers cannot effectively guide a class through it because the students cannot all view the same page in the same level, they will not recommend the site to their peers within the school or beyond.

With this in mind, we sought to create the best possible vehicle for the Foundation's information. In addition to the login redesign and major navigational enhancements, each tweak of the user interface or reordering of site information hopefully made all users more likely to log into the site, complete the game start to finish, and spread the information further in the future.

The team's experience highlighted the most important factor in ensuring the success of any awareness-aimed activism: carrying the message in the most spreadable and universally understood way possible. As an audience, website users require particular careful accommodation, relative to readers or movie watchers, as the Foundation sometimes addresses. However, the problems the learning game encountered and the technical team's solutions could inform the Foundation or other activists well in any medium. By prioritizing the active diffusion

of their message among everyone they reach, hopefully Amazon Aid can help guide us towards a political and cultural climate where the rainforest's resources are well respected and protected.

6. Future Work

As we work to spread awareness for the Amazon Rainforest, we want to ensure that the site is as smooth and simple as possible to get the job done. Amazon Aid now has a functioning learning module for teachers to easily administer to their students. However, there are still a few things we would recommend doing to perfect it over the next year.

First, there are minigames that users are encouraged to play during the "Grow A Tree" modules. While they are currently imported from a third party source, we would advise that future developers instead create a few simple games integrated directly into the site. This would work better because Amazon Aid would no longer have to pay a subscription for the third party, and the minigames could actually be interactive within the site itself. Students' scores on games could be compared amongst each other, and it could create a more enjoyable and entertaining environment for younger kids.

Another addition would be to have email authentication and notifications for teachers. For example, when one of their students finishes a topic, an email could be sent to notify that student's teacher. Specific preferences could be set within the Settings tab on the site. As for authentication, a simple email could be sent upon registration that the teacher could use to confirm it is indeed his or her email address.

Lastly, the images were previously stored in a google drive folder before we took over the project. We recommend that in the future, all images and media for the site be stored in an S3 bucket through AWS. This is a much more common, safe, and efficient way to store those files. The client has understood this and is expecting to use S3 instead. These changes to the site are

everything that we've decided are necessary for this learning module to work to its full potential, but we expect to see success even before these changes are made.

7. References

Amazon Aid. (2020). *Homepage - Amazon Aid Foundation*. Retrieved from https://amazonaid.org/

ARCAmazon. (2020). *ARCAmazon | Alliance for Research and Conservation in the Amazon Rainforest in Peru*. Retrieved from https://conservetheamazon.org/

Lovejoy, T. E., & Nobre, C. (2018). Amazon Tipping Point. *Science Advances*, 4(2), 1-2. https://doi.org/10.1126/sciadv.aat2340

Pereira, E. J. de A. L., Ferreira, P. J. S., Ribeiro, L. C. de S., Carvalho, T. S., & Pereira, H. B. de B. (2019). Policy in Brazil (2016–2019) threaten conservation of the Amazon rainforest. *Environmental Science & Policy*, 100, 8–12. doi: 10.1016/j.envsci.2019.06.001

Rainforest Alliance. (2020). *About Us | Rainforest Alliance*. Retrieved from https://www.rainforest-alliance.org/about

Reuters. (2019, August 29). *Surge in young Republicans worried about the environment: survey*. Retrieved from https://www.reuters.com/article/us-environment-poll-republicans/surge-in-young-republicans-worried-about-the-environment-survey-idUSKCN1VJ17V